Querying Semi-Structured Data Transformations Using Data Fusion Techniques

V. David Martin, D. Shanmugasundaram

Abstract— The development of the Internet in recent years has made it possible and useful to access many different information systems anywhere in the world to obtain information. While there is much research on the integration of heterogeneous information systems, most commercial systems stop short of the actual integration of available data. Here the semi-structured data and their transformations to be verified and enhanced by using the fusion techniques.

Index Terms— AUG Signals, CFAR, Lindexes, OQL, Transformation Algorithm, , Vindexes

1 Introduction

Data fusion is the process of fusing multiple records representing the same real-world object into a single, consistent, and clean representation

Traditional database systems force all data to adhere to an explicitly specified, rigid schema. For many new database applications there can be two significant drawbacks to this approach. The data may be irregular and thus not conform to a rigid schema. In relational systems, null values typically are used when data is irregular, a well-known headache. While complex types and inheritance in object-oriented databases clearly enable more flexibility, it can still be difficult to design an appropriate object-oriented schema to accommodate irregular data.

It may be difficult to decide in advance on a single, correct schema. The structure of the data may evolve rapidly, data elements may change types, or data not conforming to the previous structure may be added. These characteristics result in frequent schema modifications, another well-known headache in traditional database systems Because of these limitations, many applications involving Semi structured data are forgoing the use of a database management system, despite the fact that many strengths of a DBMS (ad-hoc queries, efficient access, concurrency control, crash recovery, security, etc.) would be very useful to those applications.

As a popular first example, consider data stored on the World-Wide Web. At a typical Web site, data is varied and irregular, and the overall structure of the site changes often. Today, very few Web sites store all of their available information in a database system. It is clear, however, that Web users could take advantage of database support, e.g., by having the ability to pose queries involving data relationships (which usually are known by the site's creators but not made explicit). As a second example, consider information integrated from multiple, heterogeneous data sources. Considerable effort is typically spent to ensure that the integrated data is well-structured and conforms to a single, uniform schema. Additional effort is required if one or more of the information sources changes, or when new sources are added. Clearly, a database system that easily accommodates irregular data and changes in structure would greatly facilitate the rapid integration of heterogeneous databases.

- ***V. David Martin, M.Sc., M.Phil., C.S.T.P.**, Lecturer in Computer Science, H.H. The Rajah's College(Auto.), Pudukkottai, Tamilnadu, India- 622001, PH-9842098741. E-mail: haidavein@gmail.com*
- ***D. Shanmugasundaram, M.Sc., M.Phil.,** Asst. Prof. of Computer Science, H.H. The Rajah's College(Auto.), Pudukkottai, Tamilnadu, India- 622001, PH-9865467456. E-mail: dshhrc@gmail.com*

2 Data Fusion Techniques

With more and more information sources available via inexpensive network connections, either over the Internet or in company intranets, the desire to access all these sources through a consistent interface has been the driving force behind much research in the field of information integration. During the last three decades many systems that try to accomplish this goal have been developed, with varying degrees of success. One of the advantages of information integration systems is that the user of such a system obtains a complete yet concise overview of all existing data without needing to access all data sources separately: complete because no object is forgotten in the result; concise because no object is represented twice and the data presented to the user is without contradiction. The latter is difficult because information about entities is stored in more than one source.

After major technical problems of connecting different data sources on different machines are solved, the biggest challenge remains: overcoming semantic heterogeneity that is, overcoming the effect of the same information being stored in different ways. The main problems are the detection of equivalent schema elements in different sources (schema matching) and the detection of equivalent object descriptions (duplicate detection) in different sources to integrate data into one single and consistent representation. However, the problem of actually integrating or fusing, the data and coping with the existing data inconsistencies is often ignored.

The problem of contradictory attribute values when integrating data from different sources is first mentioned by Dayal [1983]. Since then, the problem has often been ignored, yet a few approaches and techniques have emerged. Many of them try to avoid the data conflicts by resolving only the uncertainty of missing values, but quite a few of them use different kinds of resolution techniques to resolve conflicts.

With this survey we introduce the inclined reader to the process of data fusion in the wider context of information integration. This process is also referred to in the literature as data merging, data consolidation, entity resolution, or finding representations/survivors. We also present and compare existing approaches to implement such a data fusion step as part of an information integration process and enable users to choose the most suitable among them for the current integration task at hand.

Data fusion techniques combine data from different sources together. The main objective of employing fusion is to produce a fused result that provides the most detailed and reliable information possible. Fusing multiple information sources together also produces a more efficient representation of the data. AUG[1] Signals has been involved in research and development in the area of data fusion for over a decade. The company has developed techniques in all three categories of data fusion

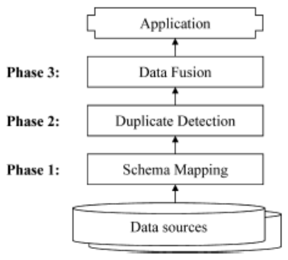


Fig. 1. A data integration process.

- ✓ **Pixel / Data level fusion**
- ✓ **Feature level fusion**
- ✓ **Decision level fusion**

Pixel level fusion is the combination of the raw data from multiple source images into a single image. Feature level fusion requires the extraction of different features from the source data –before features are merged together. Decision level fusion combines the results from multiple algorithms to yield a final fused decision. AUG Signals' fusion algorithms have been applied to various types of data including:

- ✓ **Multi-sensor data**
- ✓ **Multi-temporal data**
- ✓ **Multi-resolution data**
- ✓ **Multi-parameter data**

The two main application areas are Image Fusion and Algorithm Fusion. Image Fusion techniques use different fusion techniques to combine multiple images into a single fused image. Algorithm Fusion techniques fuse the decision results from multiple algorithms to yield a more accurate decision.

2.1 Algorithm Fusion

Algorithm fusion is an unique research area in which AUG Signals has been heavily involved. Algorithm fusion uses sophisticated rules to combine decisions from multiple algorithms into a final decision, increasing the overall performance of the system. Two Algorithm Fusion techniques are discussed below, Multi-CFAR[2] Detection and Decision Fusion of Separate Data-mining Subsystems on Multiple Data Sources.

2.1.1 Multi-CFAR Detection

Unlike single CFAR detectors, AUG Signals' Multi-CFAR detector uses several CFAR detectors, such as the Ordered Statistics (OS) CFAR detector and the Cell Averaging (CA) CFAR detector, to perform detection on the same data. The detection decisions from each detector are fused using specific rules to obtain a final detection decision. The combination of CFAR detectors is able to provide complementary information and achieve higher detection performance than any single detector, while maintaining a constant false alarm rate. Please see *AUG Signals' Technical Brief on CFAR Detection* for more details.

2.1.2 Decision Fusion of Separate Data-mining Subsystems on Multible Data Sources

The aim of fusing the decisions of separate data-mining subsystems operating on separate data sources is to increase the overall performance. Decision level fusion was chosen against data fusion and feature fusion in the three-level fusion hierarchy, because of its feasibility, lower computational complexity and robustness to the removal or addition of individual

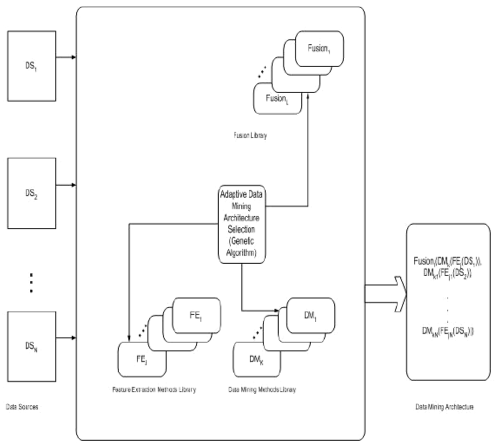
[1] AUG Signals – Airbone Underwater Geophysical Signals

2 CFAR – Constant Fault Alarm Rate

data sources. Decision fusion is the major component in the multi-source data-mining system developed by AUG Signals for decision support and situation assessment. It is able to automatically generate a solution given a library of features, data-mining algorithms and fusion techniques that are comparable to a tuned solution for the data set. The advantages of this system are its ability to incorporate:

- ***Multiple-source (sensor) data***
- ***Multiple similar and dissimilar features***
- ***Multiple data-mining algorithms***
- ***Multiple fusion methodologies***

This produces a system that exploits the resources of truth data, feature extraction, data-mining and fusion, while minimizing the level of expertise required for the end-user. The architecture for the system is shown below.


DS_1
DS_2
DS_N
Data Sources
$Fusion_1$
$Fusion_L$
Fusion Library
Adaptive Data Mining Architecture Selection (Genetic Algorithm)
FE_1
FE_J
Feature Extraction Methods Library
DM_1
DM_K
Data Mining Methods Library
$Fusion_l(DM_k(FE_j(DS_1)), DM_{k1}(FE_{j1}(DS_2)), \ldots DM_{kN}(FE_{jN}(DS_N)))$
Data Mining Architecture

Multiple Source Data-mining Architecture

2.2 Data Transformation

Integrated information systems must usually deal with heterogeneous schemata. In order to present to the user query results in a single unified schema, the schematic heterogeneities must be bridged. Data from the data sources must be transformed to conform to the global schema of the integrated information system. Two approaches are common to bridge heterogeneity and thus specify data transformation: schema integration and schema mapping. The former approach is driven by the desire to integrate a known set of data sources. Schema integration regards the individual schemata and tries to generate a new schema that is complete and correct with respect to the source schemata, that is minimal, and that is understandable

The latter approach, schema mapping, assumes a given target schema; that is, it is driven by the need to include a set of sources in a given integrated information system. A set of correspondences between elements of a source schema and elements of the global schema are generated to specify how data is to be transformed A particularly interesting addition to schema mapping are *schema matching*[3] techniques, which semi-automatically find correspondences between two schemata. There is much ongoing research in both the areas of schema matching and schemas mapping comprehensive surveys are yet missing.

The goal of both approaches, schema integration and schema mapping, is the same: transform data of the sources so that it conforms to a common global schema. Given a schema mapping, either to an integrated or to a new schema, finding such a complete and correct transformation is a considerable problem. The data transformation itself, once found, can be performed offline, for instance, as an ETL process for data warehouses; or online, for instance, in virtually integrated federated databases. After this step in the data integration process all objects of a certain type are represented homogeneously.

2.3 Duplicate Detection

The next step of the data integration process is that of duplicate detection (also known as record linkage, object identification, reference reconciliation, and many others). The goal of this step is to identify multiple representations of the same real-world object: the basic input to data fusion.

In principle, duplicate detection is simple: Compare each pair of objects using a similarity measure and apply a threshold. If a pair is more similar than the given threshold it is declared a duplicate. In reality there are two main difficulties to be solved: *effectiveness* and *efficiency*.

Effectiveness is mostly affected by the quality of the similarity measure and the choice of a similarity threshold. A similarity measure is a function determining the similarity of two objects. Usually the similarity measure is domain-specific, for instance, designed to find duplicate customer entries. Domain-independent similarity measures usually rely on string-distance measures, such as the Levenshtein-distance [Levenshtein 1965]. The similarity threshold determines when two ob-

[3] There are two other fields in computer science that also use the term *(data) fusion*. In information retrieval it means the combination of search results of different search engines into one single ranking, therefore it is also called *rank merging*. In networking it means the combination of data from a network of sensors to infer high-level knowledge, therefore also called *sensor fusion*. Beyond computer science, in market research, the term data fusion is used when referring to the process of combining two datasets on different, similar, but not identical objects that overlap in their descriptions.

jects are duplicates. A too-low threshold will produce a high recall (all duplicates are found) but a low precision (many non-duplicate pairs are declared duplicates). A too-high threshold results in high precision, but low recall. Tuning the threshold is difficult and very domain- and even dataset-specific.

Efficiency is an issue because datasets are often very large, so even calculating and storing all pairs of objects can become an obstacle. Another obstacle of efficient duplicate detection is the complexity of the similarity measure itself, for instance, because the expensive Levenshtein-distance (or edit-distance) is part of the similarity function. The first obstacle is overcome by an intelligent partitioning of the objects and comparison of pairs of objects only within a partition. A prominent example of this technique is the sorted neighborhood method [Hern´andez and Stolfo 1998]. The second obstacle can be alleviated somewhat by efficiently computing upper bounds of the similarity and computing the actual distance only for pairs whose bounds are higher than the upper bound [Weis and Naumann 2004].

The result of the duplicate detection step is the assignment of an object-ID to each representation. Two representations with the same object-ID indicate duplicates. Note that more than two representations can share the same object-ID, thus forming ***duplicate clusters***. It is the goal of data fusion to fuse these multiple representations into a single one.

2.4 Complete and Concise Data Integration

Data integration has two broad goals: increasing the completeness and increasing the conciseness of data that is available to users and applications. An increase in completeness is achieved by adding more data sources (more objects, more attributes describing objects) to the system. An increase in conciseness is achieved by removing redundant data, by fusing duplicate entries and merging common attributes into one.

In analogy to the well-known precision/recall measure from information retrieval, we can similarly define a measure of conciseness/completeness when regarding unique and additional object representations in the considered universe and the concrete data set that is being measured (see Figure).

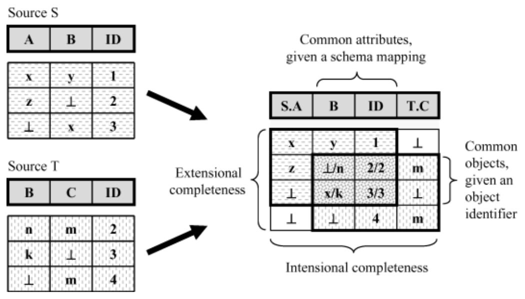


Fig. Visualizing extensional and intensional completeness when integrating two data sources (source *S* and *T* with their respective schemas (*A*, *B*, *ID*) and (*B*, *C*, *ID*)) into one integrated result, using information from schema matching (common attributes, here identified by same name) and duplicate detection (common objects, here identified by same values in the ID column).

Completeness. In analogy to recall, completeness of a dataset, such as a query result or a source table, measures the amount of data in that set, both in terms of the number of tuples (extensional, data level) and the number of attributes (intensional, schema level).

Extensional completeness is the number of unique object representations in a dataset in relation to the overall number of unique objects in the real world, such as, in all the sources of an integrated system. It measures the percentage of real-world objects covered by that dataset. We assume we are able to identify same real-world objects, for example, by an identifier created during duplicate detection.

$$\text{Extensional completeness} = \frac{\text{unique objects in data set}}{\text{all unique objects in universe}}$$

$$= \frac{a}{a+c}$$

Intensional completeness is the number of unique attributes in a dataset in relation to the overall number of unique attributes available. An increase is achieved by integrating sources that supply additional attributes to the relation; that is, additional attributes that could not be included in one of the schema mappings between the sources considered so far.

Conciseness. In analogy to precision, conciseness measures the uniqueness of object representations in a dataset. In the terms of the above Figure, ***extensional conciseness*** is the number of unique objects in a dataset in relation to the overall number of object representations in the dataset.

$$\text{Extensional conciseness} = \frac{\text{unique objects in data set}}{\text{all objects in data set}}$$

$$= \frac{a}{a+b}$$

Similarly, ***intensional conciseness*** measures the number of unique attributes of a dataset in relation to the overall number of attributes. For these measures to be useful, the definition of objects in the universe and the considered dataset needs to be the same. Schema-altering operations, such as joins, may cause problems here.

2.5 Data Fusion Answers

The result of a query to an integrated information system is called an *answer*. Such an answer could be characterized, among others, as being one of the following.

- ***Complete.*** A complete answer contains all the objects (extensionally complete) and also all attributes intensionally complete) that have been present in the sources. A complete answer is not necessarily concise, as it may contain objects or attributes more than just once.

- ***Concise.*** An answer is concise if all real-world objects (extensionally concise) and all semantically equivalent attributes (intensionally concise) present are described only once.
- ***Consistent.*** A consistent answer contains all tuples from the sources that are consistent with respect to a specified set of integrity constraints (inclusion or functional dependencies) [Arenas et al. 1999; Fuxman et al. 2005a]. In this sense, such an answer is not necessarily complete, as all inconsistent object representations are left out of the result. However, given that one of the integrity constraints is a key constraint on some real-world identifier, a consistent answer is extensionally concise for all included object representations.
- ***Complete and Consistent.*** A complete and consistent answer combines the advantages of completeness and conciseness and consists of all real world object descriptions from the sources, additionally fulfilling a key constraint on some real-world ID. Such a result contains all attributes from the sources and combines semantically equivalent ones into only one attribute. That way, intensional as well as extensional complete and conciseness is given.

The data fusion step in data integration aims at producing a complete and consistent answer.

3 Relational Operators and Techniques for Data Fusion

This section introduces standard and advanced relational operators and examines their abilities in fusing data from different data sources. As we specifically regard relational techniques for the remainder of this section, data is integrated from source tables (possibly coming from different data sources) into one integrated table. We first consider standard operators, such as union and join. Join-based techniques generally combine tuples from several tables while evaluating some predicates on some of their columns. Union-based techniques generally build a common schema first and then append the different tuple sets from the source tables. We then regard more advanced relational techniques which combine standard operators, invent a new one, or use a different approach. The descriptions of operators and techniques assume binary operators, operating on two tables only. The extension to more than two tables is straightforward in all cases; however, associativity is not always given.

Properties and Characteristics of Operators and Techniques

We describe the following operators and techniques using one or more of the following characteristics.

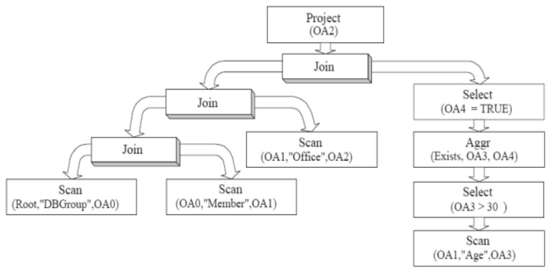


Figure 3: Example Lore query plan

OA0 (DBGroup)	OA1 (OA0.Member)	OA2 (OA1.Office)	OA3 (OA1.Age)	OA4 (true/false)

Value preservation. When combining tables in order to increase extensional completeness, not necessarily all values from the sources are included in the result (e.g., a regular join that only contains tuples from one table, if it has a join partner in the other table). However, for an operator to be fully ***value preserving,*** exactly all values for all attributes of all described objects need to remain in the result. We denote an operator that does not lose any such value or creates or duplicates values value preserving. Please note that the sole existence of a value is not sufficient and that we allow duplicate values. Thus, bag union would be an example of a value preserving operator, set union and Cartesian product examples for ***non value-preserving*** operators. Value preservation becomes important when applying (duplicate-sensitive) functions afterwards.

Value preservation should not be mistaken with the property of restorability, meaning that we can invert the operation and infer the values in the sources from the result (e.g., not possible for union). A simpler case is ***object preservation*****:** Actual values of attributes describing objects may get lost, but at least all objects that have been described in the sources are also included in the result. Objects are thereby identified by a real-world identifier. Please note that we do not consider information preservation in terms of intensional completeness

Uniqueness. An operator has the ***uniqueness preservation*** property if corresponding attributes (according to the schema mapping) that contain unique values in the sources also contain unique values in the result. This is, for example, true for an equality join on these attributes, but not for a union. ***Uniqueness-preserving*** operations are useful in scenarios where multiple representations of one real-world object only exist in different data sources (no intra source duplicates) and the unique attribute is a real world identifier. In contrast, an operator that creates a result that contains an attribute that is unique (regardless if there have been unique attributes in the sources) is said to have the ***uniqueness-enforcing*** property. In order to produce a complete and concise answer, as is the goal in data fusion, an operator needs to be uniqueness enforcing in an attribute holding a real-world identifier.

Join Approaches

Join approaches in general increase intensional completeness, as attributes from different sources are seperately included in the result. However, this comes at the cost of not guaranteeing extensionally completeness, except for the case of a full outer join. Concerning extensional conciseness, join approaches perform very well, but only if we can rely on globally unique identifiers (e.g., created by duplicate detection), and no intra-source duplicates are present.

Standard Joins. An equi-join ($\bowtie_=$) combines tuples from two relations if the join condition consists of equality conditions between columns and if it evaluates to true [Ullman et al. 2001]. The result also includes all other attributes from the sources in the result.

The Natural Join ($\bowtie$) combines two tuples from two relations if all common attributes (same name) coincide in their attribute values [Ullman et al. 2001]. Attributes that are not present in all relations are not considered in joining the relations, but are included in the result. Natural joins are also uniqueness preserving, but not necessarily value/object preserving.

The Full outer Join ($|\bowtie|$) extends the result of a standard join operation by adding tuples that are only included in one of the two source relations. The missing attributes present in only one relation are padded by null values [Ullman et al. 2001].

Full Disjunction. As the outer join operator in general is not associative, combining more than two tables by an outer join may result in different result tables, depending on the order in which the tables are joined. The full disjunction operator is defined in Galindo-Legaria [1994] as the combination of two or more tables, where all matching tuples are combined into one single tuple. It could therefore be seen as a generalization of a full outer join to more than two relations. Full disjunction can generally not be computed by a combination of outer joins alone [Galindo-Legaria 1994]. Rajaraman and Ullman [1996] cover full disjunctions in more detail and recent work on the topic provides a fast implementation [Cohen and Sagiv 2005]. Full disjunctions are uniqueness preserving and value/object preserving, like full outer joins.

Union Approaches

Union approaches are the natural way of accomplishing extensional completeness, as the result naturally includes all tuples from both relations. In contrast to join approaches, union approaches do not perform as well concerning conciseness.

Union, Outer Union, and Minimum Union. The union ($\cup$) operator (with set semantics) combines the tuples of two union-compatible relations and removes exact duplicates, that is, tuples that coincide in all attribute values [Ullman et al. 2001]. Two relations are union compatible if they are defined on the same number of attributes and if the data types of the attributes match. Union is not defined on nonunion-compatible relations. As the two example data sources are not unioncompatible (nonmatching data types), they cannot be integrated using union. Union is not uniqueness preserving as only exact duplicates are removed, resulting in contradicting tuples, with the same key value in the source relations appearing multiple times in the result. Uncertainties and contradictions are ignored; the operator is not value- but object preserving.

Outer union ($\uplus$) overcomes one restriction of union and combines two nonunion compatible relations by first padding attributes that are not in common with null values and then performing a regular union [Galindo-Legaria 1994]. Like union, outer union is not value- but object preserving. It is not uniqueness preserving, and conflicts are ignored. Outer union is not a standard operator but can be rewritten in SQL.

The minimum union operator ($\oplus$) is defined by Galindo-Legaria [1994] as the result of an outer union from which subsumed tuples have been removed.

3.1 Other Techniques

The following techniques are neither join, nor union-based, many incorporating additional information, extending the relational model or existing relational operators, or combining operators in order to fuse data.

Considering All Possibilities. Another possible way of dealing with uncertain data is to explicitly represent uncertainty in relations and use it to answer queries. The approaches described in Lim et al. [1994] and DeMichiel [1989] add an additional column to each table, its value helping to decide whether a tuple should be included in the query answer. All these approaches implement the Consider All Possibilities strategy, using the additional information to allow the user to choose consciously among all possibilities or presenting the most probable value.

Considering Only Consistent Possibilities. A new line of research was initiated by work started in Arenas et al. [1999], which introduced the idea of returning only consistent information (not containing conflicts, given some constraints) from a database to a user when issuing a query.

Among the relational techniques, union approaches are in principle well suited for data fusion, as all information from the source tables are retained and extensional as well as intensional completeness are easily reached. Only unnecessary information (i.e., duplicate tuples or subsumed tuples) is deleted, increasing conciseness. As attributes describing the same semantic concept can be mapped, intensional conciseness is also reached in an intuitive manner. However, in most cases, too many tuples and therefore too much information is kept, requiring an additional step towards a more satisfying level of conciseness. This finally results in one single tuple per real-world object. The intuitive way of increasing conciseness together with the union approach is to apply grouping and aggregation.

On the other hand, join approaches generally retain information on same real-world objects in different columns (not in different rows). Columns with equal semantic attributes do not automatically coincide as in the union approaches. They need to be combined explicitly. Moreover, completeness comes more or less without cost, but conciseness needs to be achieved by an additional processing of the join result. If there are no intrasource duplicates in the tables, a simple combination of two rows using a user-defined function is sufficient. However, if there are intrasource duplicates, again, grouping and aggregation or some advanced join operator is needed.

4 Data Transformation

Information from theWeb has already become of major importance in helping individuals and companies to follow the current development inmany areas, analyzing market developments and making business decisions. In an online bookshop, a data warehouse, for example, can be used to manage business transaction data, such as customer orders and promotions.

The implementation of OLAP on the data warehouse will help to gain an insight into customer behavior, perform buy and replenish analysis, and design focused promotions.

However, in order to analyze market trends and make new business plans, a company's own data is not sufficient, the bookshop manager also needs information from his suppliers, partners, and about his competitors. For example, discount book information or information about new publications from his competitors is important to help him

4.1 Transformation Algorithm

After mapping rules are specified, a parser will parse the mapping rules and produce the traversing route list and a list of subtrees needed in constructing the tree representing the relational table. This data provides the input parameters of the mapping algorithm. The input tree in the algorithm can be any MIX object tree of the corresponding concepts. The output trees will be warehouse tables. The key part of the transformation algorithm is a procedure for recursively traversing an object tree from the root to the leaves until needed subtrees are reached.

At first, the transformation processor does a breadth-first traverse on a MIX object tree. If a node label matches the route information from the mapping rules, the search will be applied on its subtrees (sub objects). If two or more node labels match with the same route node label, the sub objects of that first MIX object matched will be searched first.

This procedure works recursively until all needed subtrees (MIX sub objects) are found. Once a needed sub tree is found it will be extracted, transformed and merged into the corresponding output tree. The transformation algorithm is outlined below:

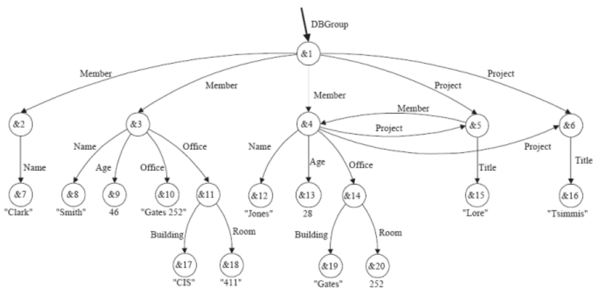
Algorithm:

Given a MIX object which can be illustrated as a tree
T, t: the root node of tree T;
P: a list of MIX objects, their ontology concepts as node labels indicate routes from the root node to the needed subtrees;
C: a list of column names of a warehouse table;
Output: a relational table which can be represented as a fixed-depth tree RT.

```
Procedure transform ( t, P, C )

Var Mark : sign of visited
m, n : nodes
p : an ontology concept in P
c : a column name in C

Begin
if t is not visited then Mark[t] : = visited;
for each m adjacent to t do

Begin
Mark[m] : = visited;
if m matches one p in P then

Begin
if p does not match any c in C then
transform (m, P, C);
else // p matches c

Begin
```

```
for each n adjacent to m do

Begin
if n is not visited then

Begin
Mark[n] : = visited;
do whatever processing is necessary on n;
merge (n, RT);

End

End

End

End

End

End
```

5 Representing and Querying Semi-Structured Data



OEM Database

The Object Exchange Model (OEM) [PGMW95] is designed for semi structured data. Data in this model can be thought of as a labeled directed graph. For example, the very small OEM database shown in Figure contains (fictitious) information about the Stanford Database Group. The vertices in the graph are objects; each object has a unique object identifier (oid), such as &5. Atomic objects have no outgoing edges and contain a value from one of the basic atomic types such as integer, real, string, gif, java, audio, etc.

All other objects may have outgoing edges and are called complex objects. Object &3 is Complex and its sub objects are &8, &9, &10, and &11. Object &7 is atomic and has value "Clark". Names are special labels that serve as aliases for objects and as entry points into the database. In OEM Database Figure, DB Group is a name that denotes object &1. Any object that cannot be accessed by a path from some name is considered to be deleted.

In an OEM database, there is no notion of fixed schema. All the schematic information is included in the labels, which may change dynamically. Thus, an OEM database is self-describing, and there is no regularity imposed on the data. The model is designed to handle incompleteness of data, as well as structure and type heterogeneity as exhibited in the example database. Observe in Figure OEM Database that, for example:

(i) members have zero, one, or more offices;
(ii) an office is sometimes a string and sometimes a complex object;
(iii) a room may be a string or an integer.

For an OEM object *X* and a label *l*, the expression *X:l* denotes the set of all l-labeled sub objects of *X*. If *X* is an atomic object, or if *l* is not an outgoing label from *X*, then *X:l* is the empty set. Such "dot expressions" are used in the query language.

5.1 The Lorel Query Language

In this subsection we introduce the Lorel query language, primarily through examples. Lorel is an extension of OQL and a full specification can be found in [AQM+96]. Here we highlight those features of the language that have an impact on the novel aspects of the system features designed specifically for handling semi structured data. Many other useful features of Lorel (some inherited from OQL and others not) that are more standard will not be covered.

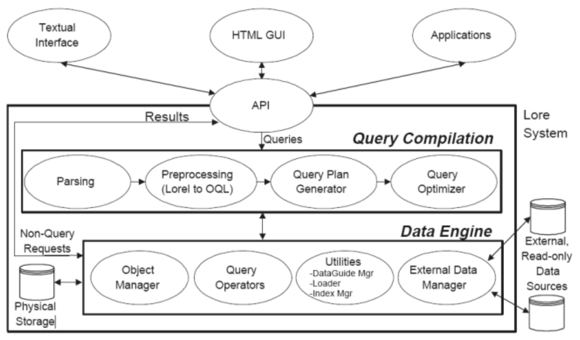
5.2 Query and Update Processing in Lore

As depicted in the above Figure, the basic steps that Lore follows when answering a query are:

- ✓ **the query is parsed;**
- ✓ **the parse tree is preprocessed and translated into an OQL-like query;**
- ✓ **a query plan is constructed;**
- ✓ **query optimization occurs; and**
- ✓ **the optimized query plan is executed**.

Query processing in Lorel is fairly conventional, with some notable exceptions: Because of the flexibility of Lorel, the preprocessing of the parse tree to produce the OQL-like query is complex.

Although the Lore engine is built around standard operators (such as Scan and Join), some take an original favor. For example, Scan may take as argument a general path expression, and therefore may entail complex searches in the database graph.



A unique feature of Lore is its automatic coercion of atomic values. Coercion has an impact on the implementation of comparators (e.g., = or <), but more importantly we shall see that it has important effects on indexing.

The result of a Lorel query is always a set of OEM objects, which become sub objects of a newly created Result object. The Result object is returned through the API. The application may then use routines provided by the API to traverse the result sub objects and display them in a suitable fashion to the user.

Because of the exibility of Lorel, the preprocessing of the parse tree to produce the OQL-like query is complex. We have implemented the specification described in [AQM+96] and we will not discuss the issue further here.

Although the Lore engine is built around standard operators (such as Scan and Join), some take an original flavor. For example, Scan may take as argument a general path expression, and therefore may entail complex searches in the database graph.

A unique feature of Lore is its automatic coercion of atomic values. Coercion has an impact on the implementation of comparators (e.g., = or <), but more importantly we shall see that it has important effects on indexing.
The result of a Lorel query is always a set of OEM objects, which become sub objects of a newly created Result object. The Result object is returned through the API. The application may then use routines provided by the API to traverse the result sub objects and display them in a suitable fashion to the user.

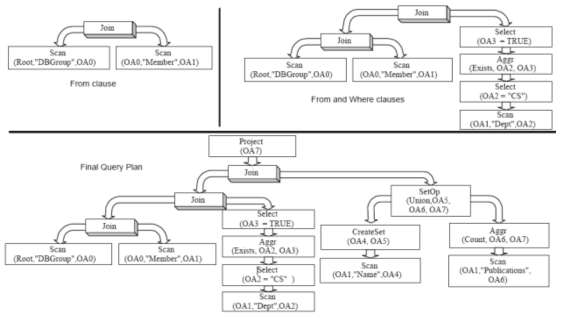
5.3 QUERY OPERATORS & PLAN

Because of the exibility of Lorel, the preprocessing of the parse tree to produce the OQL-like query is complex. We have implemented the specification described in [AQM+96] and we will not discuss the issue further here.

Although the Lore engine is built around standard operators (such as Scan and Join), some take an original flavor. For example, Scan may take as argument a general path expression, and therefore may entail complex searches in the database graph.

A unique feature of Lore is its automatic coercion of atomic values. Coercion has an impact on the implementation of comparators (e.g., = or <), but more importantly we shall see that it has important effects on indexing.

The result of a Lorel query is always a set of OEM objects, which become sub objects of a newly created Result object. The Result object is returned through the API. The application may then use routines provided by the API to traverse the result sub objects and display them in a suitable fashion to the user.



Steps of Constructing a Query Plan

To see the construction of the query plan, refer to the above Figure the sub tree for the from clause is constructed first. Each simple path expression (or range variable) appearing within the from becomes a Scan node. If several of these exist, then a left-deep tree of Scan nodes with Join nodes connecting them is constructed. At the top of the sub tree a Join node connects the *from clause* with the sub tree for the *where clause*. For where, each exists becomes a *Select, Aggr, and Scan node, and each predicate* becomes a Select node. Finally, for the *select clause*, another *Join node* is added to the top of the tree, and the query plan sub tree for the select clause becomes the right child. Let us further consider the sub tree for the *select clause*. The plans for the two expressions constituting the select clause are combined via union (using the *SetOp* operator).

Thus, each (complex) object in the result contains the set of all Name sub objects of a Member (the left sub tree of the Union), together with the count of all publications for that member. (In Lorel, a select list indicates union, while ordered pairs would be achieved using a tuple constructor operator.) The *CreateSet* operator, described earlier, is needed to obtain all Name children of a given member before returning its object assignment up the query tree. A *CreateSet* operator is not used in the right sub tree, however, since the Aggregation operator by definition already calls its subquery to exhaustion (and then applies the aggregation operator, in this case count) before continuing.

5.4 Query Optimization and Indexing

The Lore query processor currently implements only a few simple heuristic query optimization techniques. For example, we do push selection operators down the query tree, and in some cases we eliminate or combine redundant operators. In the future, we plan to consider additional heuristic optimizations, as well as the possibility of truly exploring the search space of feasible plans.

Despite the lack of sophisticated query optimization, Lore does explore query plans that use indexes when feasible. In a traditional relational DBMS, an index is created on an attribute in order to locate tuples with particular attribute values quickly. In Lore, such a value index alone is not sufficient, since the path to an object is as important as the value of the object. Thus, we have two kinds of indexes in Lore: a ***link (edge) index, or Lindex, and a value index, or Vindex.*** A ***Lindex*** takes an oid and a label, and returns the oids of all parents via the specified label. (If the label is omitted all parents are returned.) The ***Lindex*** essentially provides **"parent pointers"**, since they are not supported by Lore's object manager. A ***Vindex*** takes a label, operator, and value. It returns all atomic objects having an incoming edge with the specified label and a value satisfying the specified operator and value (e.g., < 5). Because ***Vindexes*** are useful for range (inequality) as well as ***point*** (equality) queries, they are implemented as B+-trees. ***Lindexes***, on the other hand, are used for single object lookups and thus are implemented using ***linear hashing.***

5.5 Value Indexes

In order to use ***Vindexes*** for comparisons, Lore must maintain three different kinds of ***Vindexes***:

(i) A String Vindex, which contains index entries for all string-based atomic values (string, HTML, URL, etc.).

(ii) A Real Vindex, which contains index entries for all numeric-based atomic values (integer and real).

(iii) A String-coerced-to-real Vindex, which contains all string values that can be coerced into an integer or real (stored as reals in the index).

5.6 Index Query Plans

If the user's query contains a comparison between a path expression and an integer, real, or string (e.g., \DBGroup. Member.Age > 30"), and the appropriate *Vindexes* and *Lindexes* exist, then a query plan that uses indexes will be generated. For simplicity, let us consider only queries in which the where clause consists of one such comparison. Query plans using indexes are different in shape from those based on Scan operators. Intuitively, index plans traverse the database bottom-up, while scan-based plans perform a top-down traversal. An index query plan first locates all objects with desired values and appropriately labeled incoming edges via the *Vindex*. A sequence of *Lindex* operations then traverses up from these objects attempting to match the full path expression in the comparison.4 Note that once we have an OA that satisfies the where clause, it may be necessary to use one or more Scan operations to find those components of the select expression that do not appear in the where clause.

A query plan using indexes is shown in Figure. This plan introduces four new query operators: Vindex, Lindex, Once, and Named Obj. The Vindex operator, which appears as the left child of the second Join operator, iteratively finds all atomic objects with value less than 30 and an incoming edge labeled Age, placing their oids in slot OA2. The Lindex operator that appears below the Once operator iteratively places into OA1 all parents of the object in OA2 via an

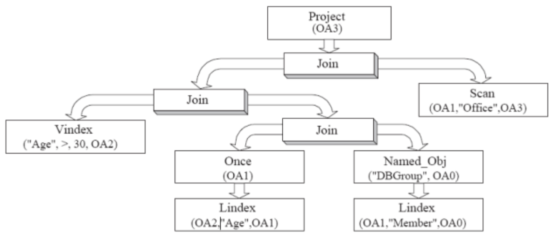


Fig. Index Query Plan

Age edge. (Since OEM data may have arbitrary graph structure, the object could potentially have several parents via Age, as well as parents via other labels.) Since Age is existentially quantified in the query, we only want to consider each parent once, even if it has several Age sub objects; this is the purpose of the Once query operator. The second Lindex operator finds all parents of the OA1 object via a Member edge, placing them in OA0. Since we want the object in OA0 to be the named object DBGroup, the Named Obj operator checks whether this is so. Once we have traversed up the database using index calls and constructed a valid OA, we finally use a Scan operator to find all Office sub objects, which are returned as the result via the topmost Project operator.

Currently, for processing where clauses, Lore only considers subplans that are completely index-based (i.e., bottom-up), such as the one discussed here, or subplans that are completely Scan-based (i.e., top-down), such as the one in the previous Figure. An interesting research topic that we have just begun to address is how to combine both bottom-up (index) and top-down (Scan) traversals. When the two traversals reach a predefined ***"meeting point"***, the intersection of the objects discovered by the index calls and the Scan operators identify paths that satisfy the where clause. The appropriate meeting point depends on the "fan-in" and "fan-out" of the vertices and labels in the database, and requires the use of statistical information.

5.7 Update Query Plans

Thanks to query plan modularity, we were able to handle arbitrary Lorel update statements by adding a single operator,

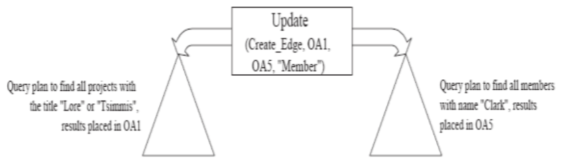

Update
(Create_Edge, OA1, OA5, "Member")
Query plan to find all projects with the title "Lore" or "Tsimmis", results placed in OA1
Query plan to find all members with name "Clark", results placed in OA5

Fig. Update Query Plan

Update, to the query execution engine. The query plan is outlined in the following Figure. The left sub tree of the Update node computes the *from* and *where* clauses of the update. In our example, the left sub tree Finds those projects with title "Lore" or "Tsimmis". For each OA returned, the right sub tree is called to evaluate the query plan for the sub- query to the right of +=. (Other valid update assignment operators are := and = [AQM+96]). In our example, the right sub tree finds those members whose name is "Clark". Once the right sub tree completes the OA, the Update node performs the actual update operation; valid operations are Create Edge, Destroy Edge, and Modify Atomic. In our example, the Update node creates an edge labeled Member between each pair of objects identified by its subtrees. Clearly a number of optimizations are possible in update processing. For instance, in our example the right sub tree of the Update node is uncorrelated with the left sub tree and thus needs to be executed only once. We currently perform this optimization, and we are investigating others.

6. CONCLUSION

In this paper we introduced the problem of data fusion in the larger context of data integration, where data fusion is the last step in a data integration process, schemata have been matched, and duplicate records have been identified. Merging these duplicate records into a single representation and at the same time resolving existing data conflicts is still out of the focus of mainstream research in the field of information integration systems. However, the problem has been addressed by several researchers in the past decades.

In the second part of this paper, we gave an overview and compared common relational techniques for data fusion. We showed how they cope with data conflicts and mention the characteristics of the results that they produce. In the third part, we presented and commented on a list of information integration systems that are capable of fusing data in various ways. We classified the systems according to their abilities of handling conflicts.

We have introduced a data-driven framework for the automatic discovery of query transformation rules in semantic query optimization. Based on this framework, we have introduced a grid representation as the data distribution in the space of a set of finite attributes. We have provided an algorithm for extracting a set of query transformation rules. We have shown the correctness of our algorithm and the rules derived from it. We also proved the completeness of our algorithm in a limited sense. Finally, an extended example of a semantic query optimization session was provided to demonstrate our discovery framework.

As part of our future research, we would like to extend our framework to the discovery of statistical rules for processing statistical queries. Lastly, we would like to investigate the benefits of data fusion and its usages in semi structured data transformations

References

[1] ADALI, S., CANDAN, K. S., PAPAKONS TANTINOU, Y., AND SUBRAHMANIAN, V. S. 1996. Query caching and optimization in distributed mediator systems. In *Proceedings of theACMInternational Conference on Management of Data SIGMOD*. ACM Press, 137–146.

[2] AGRAWAL, P., BENJELLOUN, O., SARMA, A. D., HAYWORTH, C., NABAR, S. U., SUGIHARA, T., AND WIDOM, J. 2006. Trio: A system for data, uncertainty, and lineage. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 1151–1154.

[3] AHMED, R., DE SMEDT, P., DU, W., KENT, W., KETABCHI, M. A., LITWIN, W. A., RAFII, A., AND SHAN, M.-C. 1991. The Pegasus heterogeneous multidatabase system. *IEEE Comput. 24*, 12, 19–27.

[4] AMBITE, J. L., ASHISH, N., BARISH, G., KNOBLOCK, C. A., MINTON, S., MODI, P. J., MUSLEA, I., PHILPOT, A., AND TEJADA, S. 1998. Ariadne: A system for constructing mediators for Internet sources. In *Proceedings of the ACM International Conference on Management of Data SIGMOD*. ACM Press, 561–563.

[5] BATINI, C., LENZERIN, M., AND NAVATHE, S. B. 1986. A comparative analysis of methodologies for database schema integration. *ACM Comput. Surv. 18*, 4, 323–364.

[6] DITTRICH, K. R. AND DOMENIG, R. 1999. Towards exploitation of the data universe: Database technology for comprehensive query services. In *Proceedings of the International Conference on Business Infromation Systems (BIS)*.

[7] EITER, T., FINK, M., GRECO, G., AND LEMBO, D. 2003. Efficient evaluation of logic programs for querying data integration systems. In *Proceedings of the International Conference on Logic Programming (ICLP)*, 163–177.

[8] FAGIN, R., KOLAITIS, P. G., AND POPA, L. 2005. Data exchange: Getting to the core. *Trans. Dat. Syst. 30*, 1, 174–210.

[9] LEVENSHTEIN, V. 1965. Binary codes capable of correcting spurious insertions and deletions of ones. *Problems Inf. Transm. 1*, 8–17.

[10] LEVY, A. Y., RAJARAMAN, A., AND ORDILLE, J. J. 1996a. Querying heterogeneous information sources using source descriptions. In *Proceedings of the International Conference on Very Large Databases (VLDB)*. Morgan Kaufmann, 251–262.

[11] TSAI, P. S. M. AND CHEN, A. L. P. 2000. Partial natural outerjoin—An operation for interoperability in a multidatabase environment. *J. Inf. Sci. Eng. 16*, 4 (Jul.), 593–617.

[12] TSENG, F. S.-C., CHEN, A. L. P., AND YANG, W.-P. 1993. Answering heterogeneous database queries with degrees of uncertainty. *Distrib. Parallel Databases 1*, 3, 281–302.

[13] ULLMAN, J. D., GARCIA-MOLINA, H., AND WIDOM, J. 2001. *Database Systems: The Complete Book*. Prentice Hall PTR.

[14] WANG, H. AND ZANIOLO, C. 2000. Using SQL to build new aggregates and extenders for object- relational systems. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, A. E. Abbadi et al., eds. Morgan Kaufmann, 166–175.

[15] J. Gafiney. Illustra's web datablade module. Technical report, Informix Corporation, February 1997. Available at http://www.informix.com/informix/corpinfo/zines/whitpprs/illuswp/dblade.htm.

[16] S. Ghandeharizadeh, R. Hull, and D. Jacobs. Heraclitus: Elevating deltas to be first class citizens in a database programming language. ACM Transactions on Database Systems, 21(3):370{426, 1996.

[17] G. Graeme. Query evaluation techniques for large databases. ACM Computing Surveys, 25(2):73{170,1993.

[18] R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In Proceedings of the Twenty Third International Conference on Very Large Data Bases, Athens, Greece, August 1997.

[19] M. Kifer, W. Kim, and Y. Sagiv. Querying objectoriented databases. In Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 393{402, San Diego, California, June 1992.

[20] D. Konopnicki and O. Shmueli. W3QS: A query system for the World Wide Web. In Proceedings of the Twenty-First International Conference on Very Large Data Bases, pages 54{65, Zurich, Switzerland, September 1995.