Combining Heterogeneous Data Sources through Query Correspondence Assertions

Ulf Leser
TU Berlin FB 13, CIS
Einsteinufer 17
D-10587 Berlin, Germany
Tel: +49 30 314 79463
leser@cs.tu-berlin.de

1. ABSTRACT

The WWW today offers free access to a wealth of heterogeneous data sources. Combining related data from different sources in a comfortable and automatic fashion is not possible. We present our approach to this problem that is based on a declarative representation of the content of heterogeneous data sources with respect to a global schema. We describe our language to express these correspondences and give the algorithm that uses them to answer global queries.

1.1 Keywords

database integration, query translation, mediation

2. INTRODUCTION

The recent years have seen a steep increase in the amount of data available for public use, mainly pushed through the success of the World-Wide-Web. However, combining data from different sources to yield more complete answers or to answer more complex queries faces many problems. For instance, the same information is often stored using different structures in different sources. Obtaining the same piece of data from different sources can reveal contradicting values. Furthermore, many source are only capable of answering a fixed set of (possibly parameterized) queries.

Mediator based integration architectures define a framework to deal with such problems [11]. In such systems, the mediator holds a schema (*mediator schema*) which semantically subsumes the interesting parts of the source schemas. Technical and syntactical heterogeneity in the sources is hidden by wrappers which offer a uniform interface to the mediator. In our approach this interface is comprised of a relational export schema (*source schema*) and the set of possible queries against this schema. The mediator tries to find answers for queries against the mediator schema by combining data from different sources which are accessed through their wrappers. In this process, many types of schematic and semantic discrepancies have to be bridged ([4]).

In this work we concentrate on the representation of semantic correspondences between the source schemas and the mediator schema and how these representations can be used to answer global queries in a meaningful, i.e. semantically correct way. We propose *query correspondence assertions* (QCA) as a flexible mechanism to express correspondences between heterogeneous schemas. With a QCA, a human administrator defines the intentional equivalence of two views, where one is defined as a query against the mediator schema (*mediator query*, MQ) and the other is defined as a query against one source schema (*source query*, SQ). Consider the mediator schema given in Fig. 1 and Tab. 1 and the sources described in Tab. 2. Source 1 permits queries that obtain tuples reporting positive experiments between a gene and a clone, together with the sequence of the gene and the type of the experiment. The same information is stored in two relations in the mediator schema. We define this correspondence with the following QCA:

```
exp(C,G,T),gene(G,S,D) =: v2(C,G,S,T) ⊇
                    S1.v2(C,G,S,T) := pos(G,S,C,T)
```

which can be understood as follows: if the wrapper executes the query on the right hand side, e.g. by filling out a form in the WWW interface of source 1, a set of tuples (G,S,C,T) is returned. The query on the left hand side determines how the values of these tuples must be sorted into relations on the mediator level.

We do not require that a user query (UQ) directly matches with the left hand site of a QCA. In general, the mediator searches for combinations of MQs (*plans*) that together obtain semantically correct answers. For instance, answering a query that exceeds the domain of one single source requires the combination of QCAs from different sources. As we assume that no source has a complete data set, there can be many correct plans.

This paper is organized as follows: section 3 defines the language in which QCAs are expressed. Section 4 describes our algorithm to find correct plans for arbitrary user queries. Section 5 discusses related work and section 6 concludes. A more detailed version of this work is [7], and some architectural thoughts on our approach can be found in [8]. Fig. 1 depicts the mediator schema we use for our examples. Tab. 1 gives the according schema,

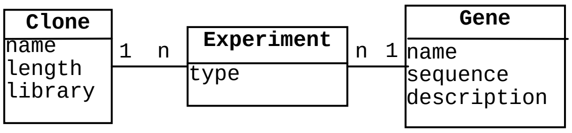

Figure 1: A simple gene mapping schema. To analyze a gene, biologists need a clone containing this gene. *Clones* are long stretches of genomic DNA which are used to determine the actual sequence of a chromosome. They are arranged in libraries. A positive experiment shows that a certain gene is contained in a certain clone.

Tab. 2 describes available sources. The resulting mediator should form an information resource for the X chromosome.

3. QUERY CORRESPONDENCE ASSERTIONS

Formally, a QCA has the form:

$$\mathrm{MQ} =: v(e_1 \ldots e_n) \supseteq s(e_1 \ldots e_n) := \mathrm{SQ}$$

where MQ is a conjunctive relational query [1] against the mediator schema and SQ is a relational query against the export schema of a wrapper which can be executed by the corresponding source. *v* (resp. *s*) is a view on the mediator (source) schema defined by MQ (SQ) which projects out variables being present in only MQ or SQ, respectively. We will prefix *s* with the source name for clarity. We call the e_i *exported variables*, and each e_i must appear both in SQ and MQ for safety. MQ and SQ can contain constants in the position of attributes, thus expressing selections.

Semantically, a QCA defines that the extension of *s* is a subset of the (only virtually existing) extension of *v*. To retrieve this subset, the mediator sends SQ to the appropriate wrapper who accesses the source, executes the query and returns records ($l_1,...,l_n$) for *s*. For each such record, a new row is created for each occurrence of each relation in MQ with values as follows: let *r* be a relation in MQ with arity *k*, let a_i denote the symbol at position i of *r* in MQ, and let ($z_1,...,z_k$) be the new row of *r*. 1) if a_i is a constant *c*, then $z_i=c$. 2) otherwise, if a_i is an exported variable, hence $\exists\ j: a_i=e_j$, then $z_i=l_j$. 3) otherwise, if a_i does not appear in any other relation $p \in MQ$, then $z_i=null$. 4) otherwise, a_i is not exported but used in a join condition. Then an artificial value is created and used in all occurrences of a_i in MQ to maintain this join.

In general, answering a UQ requires a combination of many QCAs, The necessary inter-QCA join conditions are automatically generated by the planning algorithm (see below). It is therefore often the case that some attributes are instantiated (through previously executed SQs) before a SQ is executed; such selections must be enforced inside the wrapper. The *result* of a plan is the union of all exported variables of the contained QCAs.

`exp(C,G,T)`	Experiments of type T found gene G in clone C.
`gene(G,S,D)`	Gene G has sequence S and description D.
`clone(C,L,Y)`	Clone C has length L and library Y.

Table 1 : The mediator schema. As we only want X chromosome data, we have no attribute for the chromosome of an object.

We give examples according to the sources described in Tab. 2. The two possible queries for S1 are captured through two QCAs:

```
R1: clone(C,L,Y) =: v1(C,L) ⊇
                        S1.v1(C,L) := cinfo(C,L)
```

```
R2: exp(C,G,T),gene(G,S,D) =: v2(C,G,S,T) ⊇
                  S1.v2(C,G,S,T) := pos(G,S,C,T)
```

R1 defines that the relations *clone* and *cinfo* are semantically equivalent in the semantic context of the mediator. This is true because we know that S1 contains only X chromosome objects. However, *cinfo* does not contain the attribute *library*; hence, if R1 would be executed, resulting tuples in *clone* would have a *null* at this position. R2 states that the values stored in *S1.pos* are distributed over two relations in the mediator and determines how this distribution looks like. S1 stores the sequence of a gene in the same relation as its experimental results. On first sight, this seems to be an awkward schema, as one gene has only one sequence, but many positive clones; however, such a representation is for instance well-suited for a WWW report that results from a form-based query. Accepting available queries as they are greatly reduced the time that has to be invested in the implementation of the wrapper.

R3 describes the content of S2:

```
R3: exp(C,G,'hyb'),gene(G,S,D) =: v3(C,G,D) ⊇
         S2.v1(C,G,D) := hyb(C,G,'X'),ginfo(G,D)
```

R3 includes conditions in the queries that are necessary to achieve semantic equivalence. From the mediator point of view, all data stemming from S2 have the experimental type ‘hyb’. Hence, queries asking for all experiments of e.g. type ‘PRC’ must not access S2. On the other hand, S2 stores results for objects of all chromosomes, while we are only interested in X chromosome data. This diverging context requires a filtering of tuples using the condition *Chr='X'*.

Finally, source 3 is modeled through R4:

```
R4: exp(C,G,T),clone(C,L,Y) =: v4(C,G,L,Y) ⊇
              S3.v1(C,G,L,Y) := res(G,C,L,Y,'X')
```

4. PLANNING WITH QCAs

In this section we describe our algorithm that, given a user query, finds all plans such that, if all QCAs in a plan are executed subsequently, the result contains all and only correct answers to the original query with respect to the given set of QCAs. The idea to the solution of this problem builds on an analogy to global optimization of conjunctive queries (see [3]). [9] shows that the problem is NP-complete and proves that a plan for a query with *n* relations will need at most *n* views, i.e. QCAs.

A straight-forward algorithm first enumerates all possible combinations of MVs. Each combination is expanded, i.e. the views are substituted by their definitions. Then it is tested whether this query is contained in UQ. A query q is contained in a query p if there is a *containment mapping* ([2]) (short: *mapping*) from p to q, i.e. a mapping from symbols of p to symbols of q with the following properties: no variable is mapped to two different

Export schema	Possible queries	Description
`S1.cinfo(C,L)` `S1.pos(G,S,C,T)`	`v1 := cinfo(C,L)` `v2 := pos(G,S,C,T)`	S1:contains only clones from the X chromosome. One relation stores experimental results plus the sequence; one stores clones and their length.
`S2.hyb(C,G,Chr)` `S2.ginfo(G,D);`	`v1 := hyb(C,G,Chr), ginfo(G,D)`	S2:contains genes from all chromosomes found in clones by hybridisation experiments.
`S3.res(G,C,L,Y,Chr);`	`v1 := res(G,C,L,Y,Chr)`	S3:contains experimental results together with the length, library and chromosome of the clone in one relation.

Table 2 : Available data sources: export schema, executable queries and description.

symbols; no exported variable is mapped to a non-exported variable; if p has a constant at position *i*, then q has the same constant at position *i*; and each predicate of p maps to a predicate of q.

We present an algorithm (alg. 1) that contains several enhancements compared to this naive approach. It incrementally constructs correct plans instead of generating and testing all possible plans. Input is a (finite) set of QCAs (*W*; we only need MQ and MV here) and a user query U, where U is of the form UV:=UQ. Let UQ consist of *n* relations $u_1...u_n$. UV is the view defined by UQ and can project out variables. The output is a set of plans, each being a sequence of QCAs. The *complete answer* to U is the union of the results of all plans.

To explain the algorithm we need some definitions:

- A predicate q *covers* [2] a predicate p if a mapping m exists that maps q to p. Note that m is unique if it exists at all. We write $q >_m p$ if q covers p under m.
- Two mappings are *c-compatible*: $m_1 \equiv_c m_2$, iff no variable is mapped to two different constants in m_1,m_2.
- Two mappings are *v-compatible*: $m_1 \equiv_v m_2$, iff $m_1 \equiv_c m_2$ and no variable is mapped to two different variables in m_1,m_2; note that m_1 and m_2 can still be defined for different variable sets.

Clearly, each correct plan must contain a predicate p for every predicate $u_i \in UQ$ with $u_i >_m p$. We reduce the test for query containment to the test for compatibility between mappings.

Our algorithm (Alg. 1) starts by calculating a vector $cover[u_i]:=\{(p,m) | p \in W,\ u_i >_m p\}$. Hence, $cover[u_i]$ contains all predicates of any QCA that are covered by u_i together with the corresponding mapping. It then enumerates all combinations $\{z_1...z_n\}$ with $z_i \in cover[u_i]$. Let *Z* be such a combination. Note that each z_i has a defined QCA from which it originates; let *org* be the function that returns this QCA, i.e if $z_i \in Q_j$, then $org(z_i)=Q_j$. *Z* represents the sequence of QCAs $<Q_1...Q_n>$, where $Q_i=org(z_i)$. However, *Z* is not yet what we want:

- *Z* is only valid if all contained mappings are c-compatible to each other, i.e. $\forall\ z_1=(p_1,m_1), z_2=(p_2,m_2),\ z_1 \in Z,\ z_2 \in Z \Rightarrow m_1 \equiv_c m_2$. Otherwise, Z will map a variable to different constants. It is not necessary that the mappings are v-compatible: if one variable from UQ is mapped to two different variables in the QCAs, then a inter-QCA join-condition will be created in the final plan.
- Z is not minimal, because each z_i is treated as if it would stem from a separate QCA, ignoring the possibility that different z_i, stemming from the same QCA, can sometime be executed by invoking this QCA only once.

```
Inp: W={Q_i}; // Q_i={MQ_i,MV_i}
     U={UV,UQ};
Out: S:={<Q_1..Q_m>|∀t∈<Q_1..Q_m>⇒t∈UV;m≤|UQ|;
foreach u∈UQ:
  cover[u]={};
  foreach u ∈ UQ
    foreach q ∈ W
      foreach p ∈ q
        if u>_m p then cover[u]:=cover[u]∪(p,m);
S:={};
foreach Z∈{<z_1...z_n>|z_i∈cover[u_i]}
  if c-compatible( m_1,...m_n) then
    Z':=minimize(Z);
    S:=S∪expand(Z');
```

Algorithm 1. Calculates *cover[u]* for each relation of the user query U. Minimal plans are generated using Alg. 2. *expand* is explained in the text

Algorithm 2 minimizes a plan by finding and removing redundant z_i. Informally, a $z_i=(p,m)$ is redundant if there is a $z_j \in Z$ with $org(z_i)=org(z_j)$ and the expansion of z_j into the full MQ will add *p* to the plan with a compatible variable mapping. Then it is not necessary to have $org(z_i)$ twice in Z. Here, c-compatibility is not enough, as z_i and z_j will be mapped to the same QCA. Note that the mapping rules for z_i need to be preserved if it is removed; we simply add it to the mapping rules of z_j.

Finally, each minimized plan is expanded. This consists of two steps. First, each z_i is substituted with $org(z_i)$. Second, the variables in $org(z_i)$ must be substituted to achieve a consistent and correct plan which contains necessary joins and identifies the exported variables of UV. For the second step, we can use the inverted mapping m^{-1} (with $z_i=(p,m)$), i.e. z_i is replaced by $m^{-1}(org(z_i))$. If two variables v_1, v_2 of UQ are mapped to the same variable v of the plan, i.e. $\exists\ v_1,v_2,m_i,m_j\!: [v_1 \rightarrow v] \in m_i$ and $[v_2 \rightarrow v] \in m_j$, then v must be substituted consistently by either v_1 or v_2. If the plan contains a variable for which no mapping exists, then this variable is not renamed.

4.1 Example

Consider the user query:

```
Inp: Z:={z_1...z_n};
Out: Z':={y_1...y_m},m≤n,∀y_i:∃z_j:y_i=z_j;
Z'={};
foreach z_i∈Z,zi=(p_1,m_1) {
  if (∃z_j=(p_2,m_2),q: z_j∈Z,org(z_j)=org(z_i)=org(q),
                 u_i >_n p, n≡_v m_1,n≡_v m_2,m_1≡_v m_2) then
    m_2:=m_2∪m_1;
  else
    Z':=Z'∪z_i;
```

Algorithm 2. Function *minimize* finds redundant elements in a plan Z.

```
q(Cl,Ge,Ty,De):=exp(Cl,Ge,Ty),gene(Ge,Se,De);
```

In step 1, we construct *cover*. To avoid confusion, we rename all variable by prefixing them with the name of the rule that they appear in:

- *cover[exp]={(R2.exp,[Cl→R2_C,Ge→R2_G,Ty→R2_T]),(R3.exp, [Cl→R3_C,Ge→R3_G,Ty→'hyb'])}*. R4 is not contained because the exported variable $Ty \in UV$ can not be mapped to an exported variable.
- *cover[gene]={(R3.gene,[Ge→R3_G,De→R3_D, Se→R3_S])}*. R2 does not contain D and is excluded.

In step 2, we enumerate different combinations and try to minimize them. All mappings are c-compatible. There are two possible plans:

1. <*R2.exp,R3.gene*>. This plan cannot be minimized because all *z* stem from different QCAs.

2. <*R3.exp,R3.gene*>: Here, $org(z_1)=org(z_2)$. Consider z_1. Then z_j ($=z_2$) and *p* (*=R3.exp*) exist as required above. This plan can hence be reduced to <*R3.gene*> with the combined mapping *[Ge→R3_G, De→R3_D, Se→R3_S, Cl→R3_C, Ty→R3_T]*.

Expansion and re-mapping yields the final plans that can be executed:

1. *S1.v2(Cl,Ge,S,Ty),S2.v1(C,Ge,De)*; here, *R2_G* and *R3_G* had the same origin (*Ge*) and form now the inter-QCA join. Note that $C \neq Cl$ (*C* has no mapping in *S2.v1* and is hence not substituted during the re-mapping); this plan tests positive experiments in S1 and only retrieves the gene description from S3, without checking whether the same positive result is present in S3. Fully considering contradicting data in different sources requires such a behavior.

2. *S2.v1(Cl,Ge,'hyb',De)*, which is in fact almost equivalent to the MQ of R3.

5. RELATED WORK

The idea of approaching the integration of heterogeneous databases as a problem of answering queries using views was, to our best knowledge, pioneered by the Information Manifold project ([10]). Our approach is similar but has some important differences: first, QCAs are more expressive than the content descriptions of the IM project, as they allow for complex queries against the sources. Although we did not expand on this aspect in this work, it adds considerable flexibility to the overall approach. Second, our algorithm explicitly treats the case that some attributes, although necessary for the specification of MQs, may not be exported by a source. Furthermore, it is not clear how the algorithm of [10] deals with the occurrence of self-joins in mediator queries.

[5] also give an algorithm that is related to ours using the generic planner OCCAM. However, they use a different semantic of queries leading to different plans.

6. DISCUSSION

We have presented a new approach to the problem of integrating heterogeneous, distributed data sources. It is based on QCAs which offer an elegant and declarative mechanism to express semantic correspondences in the presence of structural differences. There are also conflicts that can not be treated: for instance, if a concept is represented as an attribute name in a source but as an attribute value in the mediator schema (see e.g. [6]).

Projects in data integration have recently been classified as either '*global-as-view*' or '*local-as-view*', meaning that either the global relations are defined as views over the source schemas or the local relations are defined as views over the global schema. We believe that QCAs combine the best from both worlds. Our planning algorithm was implemented in JAVA. Current research focuses on multi-query optimization in SQs and the further study of redundancy in query plans.

7. ACKNOWLEDGMENTS

We are grateful to A. Bergholz, R. Kutsche and J.C. Freytag for helpful comments. This work was supported by the German Research Society grant GRK 316.

8. REFERENCES

[1] Abiteboul. S. , R. Hull, and V. Vianu, *Foundations of Databases*. Addison-Wesley, 1995.

[2] Aho, A. V., Y. Sagiv, and J. D. Ullman, "Efficient Optimization of a Class of Relational Expressions," *ACM TODS*, vol. 4, pp. 435-454, 1979.

[3] Chandra, A.K. and P.M. Merlin, "Optimal Implementation of Conjunctive Queries in Relational Databases", 9th ACM Symp. on Theory of Computing, 1977.

[4] Kim, W., I. Choi, S. Gala, and M. Scheevel, "On Resolving Schematic Heterogeneity in Multidatabase Systems", in *Modern Database Systems*, W. Kim, Ed., ACM Press, Addison-Wesley, 1995.

[5] Kwok C. T. and D. S. Weld, "Planning to Gather Information," University of Washington, Technical Report UW-CSE-96-01-04, 1996.

[6] Lakshmanan, L. V. S., F.Sadri, I. N. Subramanian, "SchemaSQL: A Language for Interoperability in relational Multidatabase Systems", 22nd VLDB, Bombay, India, pp. 239 - 250, 1996

[7] Leser, U., "Combining Heterogeneous Data Sources through Query Correspondence Assertions" extended version, available from the author, 1998

[8] Leser, U., "Maintenance and Mediation in Database Federations", WITS, Helsinki, to appear.

[9] Levy, A.Y., A. O. Mendelzon, Y. Sagiv, and D. Srivastava, "Answering Queries using Views", 14th ACM PODS, San Jose, CA, 1995.

[10] Levy, A. Y., A. Rajaraman, and J. J. Ordille, "Querying Heterogeneous Information Sources Using Source Descriptions," 22th VLDB, India, 1996.

[11] Wiederhold, G., "Mediators in the architecture of future information systems", IEEE Computer 15(3):38-49, 1992