Query Mediation for Heterogeneous Data Sources

Ulf Leser

Technische Universität Berlin,
Fachbereich 13 - CIS,
Einsteinufer 17, D-10587 Berlin.
Email: leser@cs.tu-berlin.de

Abstract:

We present a novel approach to the problem of query mediation in a tight federation of heterogeneous, distributed data sources. It is based on query correspondence assertions (QCAs), which are set-equation between conjunctive queries. We use them as rules to express relationships between heterogeneous and autonomously maintained schemas. We describe an algorithm that uses QCAs to translate queries against the global schema in semantically meaningful and minimal sequences of queries against data sources. A salient feature of our approach is that it is, due to the declarativeness of QCAs, relatively easy to react on schema changes in sources or on the global level. This supports maintainability, which we regard as a key requirement for large-scale projects in information integration.

1. 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. For instance, in the field of molecular biology there currently exist more than 400 databases which can be accessed freely by any researcher ([12]). However, these databases can be relational, object-oriented or just flatfiles; some are accessible through a high-level query language, others offer form-based WWW access or have no query facilities at all[1]. In contrast to many other domains, such as environmental information systems, sources are highly overlapping on the schema level and on the instance level. For instance, there are many databases storing information about genes, and there is no strict rule which of those contains a particular gene, but it frequently happens that objects appear in many sources. In such cases, i.e. if information about the same real world object is found in different databases, values are often contradicting to each other. This is mainly due to the fact that most of the data is generated through 'wet' experiments that come along with a certain degree of fuzziness in the results ([18]).

Getting an exhaustive overview of the available data in a certain domain or for a certain genomic object is therefore a tedious and error-prone task. It first requires searching for potential data sources, for instance by browsing the WWW. Next, the user need to understand the interfaces and query possibilities of relevant sources. After submitting a query, the representation format of the result has to be recognised. It is also frequently the case that queries can not be answered by accessing one single source, but data from many sources has

[1] Imagine a data source that consist only of a large HTML table.

to be combined, requiring the formulation of different queries. This currently has to be done manually.

In contrast to the current situation, biologists would rather like to have a single interface that offers integrated access to diverging sources. This encompasses schema, location and query language transparency. Users do not want to bother about where to find data, nor how the data is formatted in different systems, nor how they can be accessed.

Our strategy is therefore based on a global schema for user queries and a uniform query language. Receiving a user query, the task of the integrating system is to:

- select relevant sources
- decompose the query into subqueries according to the content of the sources
- translate each subquery into a request that can be answered by the source
- submit the translated subqueries and
- collect and merge the results.

The last decade has seen a number of projects addressing the integration of semantically and schematically heterogeneous databases (for instance [26, 23, 13, 10]). One important issue in this area that in our mind was not yet considered sufficiently is *maintainability*. Maintainability appears in various aspects. First, the global level should be protected against changes inside the sources as much as possible, enabling their independent evolution. Second, taken part in a integrated system should impose as few as possible requirements to a source. If more than a handful of sources are to be integrated, it is vital to make addition or deletion of data sources easy.

On the other hand, classes and objects in data sources must be clearly defined in their relation to the global schema. It is necessary to represent even fine-grained differences if a semantically correct translation of queries and integration of results should be guaranteed. We conclude that the mechanism for describing the data in the sources with respect to the global schema is of outstanding importance both for maintenance and semantic reliability.

In this work we present *query correspondence assertions* (QCA) as a novel approach to the representation of inter-schema correspondences. Our general architecture for information integration is based on a generic wrapper-mediator architecture ([29]). QCAs are therein used to define semantic relationships between queries against data sources and queries against the global schema. Mediators use these assertions to generate sequences of queries against source databases that together yield the answer to arbitrary user queries. Our approach is therefore declarative on the sense that QCAs are rules that are interpreted by an algorithm which is identical for each mediator.

This paper is organised as follows: in section 2 we give an overview of the architecture of our information integration system. Section 3 introduces query correspondence assertions as a flexible way to relate semantically diverging schemas. Section 4 describes how these assertions are used during query planning and sketches possibilities for further query optimisation. Section 5 discusses related work, and section 6 concludes.

2. Overview

The main components of our architecture are wrappers, which are source-specific, and mediators, which offer integrated access to a set of sources using these wrappers (Fig. 1). A similar approach is taken by many other projects ([3, 22, 25]) for its advantages concerning the modularization of the entire system. We use the relational data model for the global schema and at the interfaces between mediators and wrappers. As there is still no generally

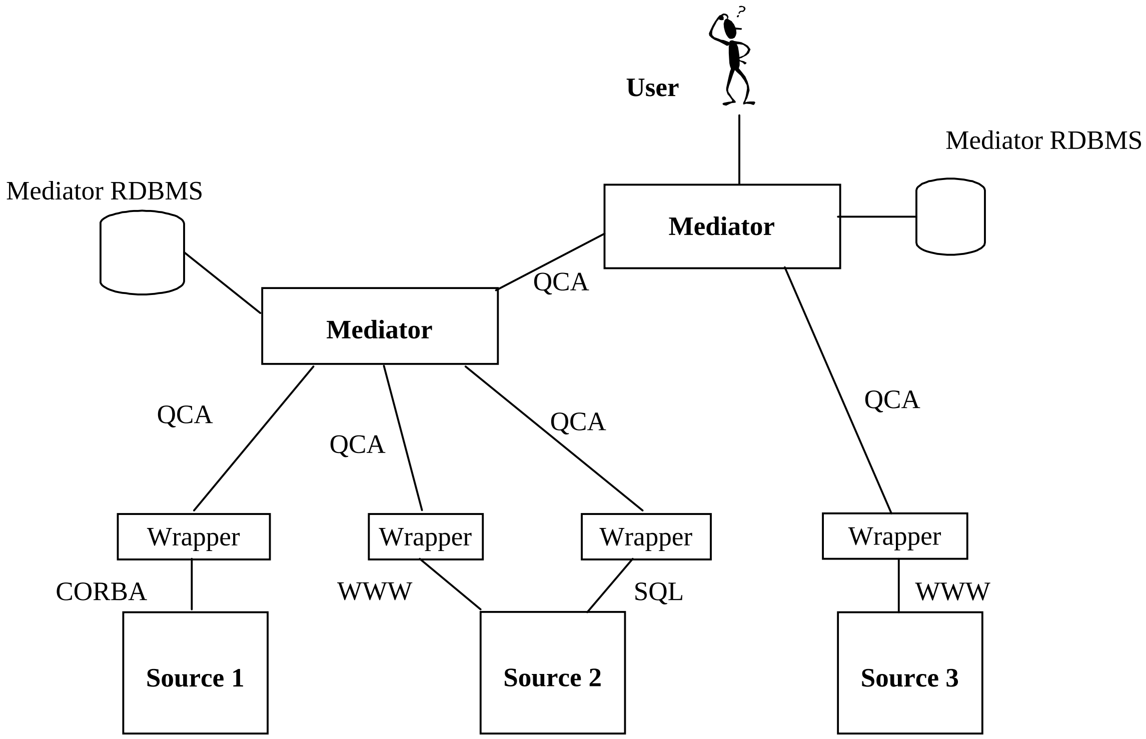

Fig.1 : Architecture overview. Mediators use wrappers to access sources. Content and capabilities of sources are defined through sets of QCAs.

accepted definition of the terms mediator and wrapper, we state more precisely how they are used in our system.

Wrapper

A wrapper is a source specific module that offers a, possibly limited, relational interface to the data contained in the source. It translates queries it receives from a mediator into requests answerable by the data source. Mediators access sources only through their wrapper. The interface between a wrapper and a mediator consists of the following parts:

- A relational export schema of the source.
- A set of queries against the export schema that are answerable by the source
- A description of the query capabilities of the source.

We do not require that the source itself is a relational database. The source interface only needs to be mapped to a relational interface, which is the task of the wrapper (see e.g. [4; 5; 9]). The wrapper hence hides the data model and the physical access mechanism, which could be HTTP, JDBC, CORBA etc. If for instance the source interface is defined by a CORBA IDL, it is often straight-forward to map the IDL interfaces to a relational schema, and to model queries through method invocations. Modelling a WWW form-based interface can be done in a similar way, i.e. the export schema is derived from the data in the source, and forms are treated as ‘canned’ queries. In case that the source is a full-fledged RDBMS, the export schema could for instance be taken directly from the data dictionary. We emphasise that a wrapper should be generated independently from the global schema, and will show in this paper how this is supported in our system.

To allow meaningful query translation, a wrapper provides a semantic mapping of its export schema into the schema of a mediator. This mapping consists of a set of rules, namely query correspondence assertions (see section 3). A wrapper can be viewed as two closely connected modules, one that hides the heterogeneity in the access mechanism and one that defines the exported data structures and their semantic with respect to a mediator schema. While the former is, in general, a procedural, hard-coded component, the latter uses a declarative mechanism. Only these specifications need to be reconsidered if a wrapper shall be used by different mediators, or if either the schema of the mediator or the export schema of the wrapper needs to be changed.

A source can have different wrappers, for instance one for a WWW interface, using pre-prepared pages for a limited number of queries which are very quick to retrieve, and one for direct SQL queries, if canned queries are not satisfying. These different wrappers can have different export schema and often support different queries.

Mediator

Every mediator has a relational schema (the *mediator schema*), modelling its perception of the world. To answer a query against this schema, it consults the QCAs of the wrappers it knows and uses them to find sequences of source queries that together yield correct answers to the original query. The planning algorithm we describe in section 4 guarantees to find all such sequences (with respect to the specified QCAs). A query plan is executed by submitting queries to the wrapper. The union of the results of each plan is the (global) answer to the query (see section 4). A mediator has no knowledge of a wrapper apart from the three components described in the previous section. Mediators are generic in the sense that the planning algorithm is the same in all mediators - only the set of QCAs and the mediator schema changes.

Mediators can also make use of other mediators, which then take the role of a wrapper. This is an important feature enabling modularization. Specialised mediators can be administered by domain experts, which removes the need to concentrate and maintain all "world knowledge" in one central place. As mediators take the role of wrappers, they use the same mechanism to specify there content with regard to the 'using' mediator. However, in this work we only consider the case of one mediator integrating a set of wrappers.

In the following, we call a query against the schema of a mediator a *user query*, no matter whether it was posed by a real human user or by another mediator. Queries generated inside the mediator and being submitted to a wrapper or another mediator are called *source queries*. We use the very simple scenario depicted in Fig. 2 for examples. The corresponding relational schema is given in Tab.1. It describes two types of entities: *STSs* are very short pieces of genomic DNA that are used as landmarks in the genome. They are identified by their primer sequence. *Clones* are long stretches of genomic DNA which are for instance used to determine the actual sequence of a chromosome. Clones are arranged in libraries and differ in their length. To find the exact position of a clone on a chromosome, biologists test which STS of

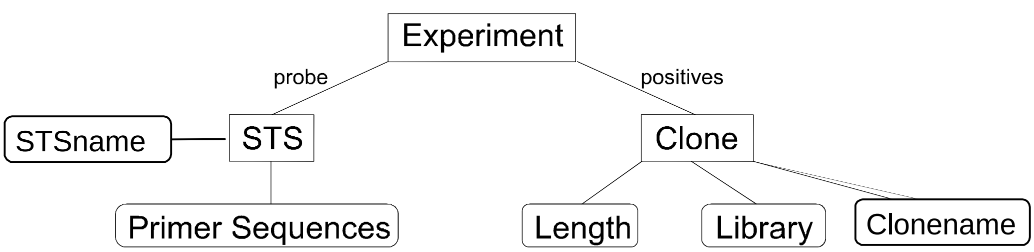

Fig. 2: A simple STS mapping schema. Entities have square corners, attributes rounded edges.

known position is contained in the clone by carrying out an *experiment* (see for instance [15] for a brief introduction in this techniques).

`exp(sts,clone)`	Relates STS with clones when an experiment has proved it..
`sinfo(sts,primer)`	STS with primer sequence.
`cinfo(clone,len,lib)`	Clones with length and library.

Tab.1 : The global schema inside the mediator.

We want to build a system that integrates all X chromosome data contained in one of the data sources described in Tab. 2.

`s1.pos(sts,clone);` `s1.ci(clone,lib);`	A WWW interface to X specific experimental results. All clones used are smaller than 150 kb. Another form takes as input a clone name and gives back its library.

`s2.sts(sts,chr);`
`s2.clone(clone,len,chr);`
`s2.exp(sts,clone,intensity);`
	A relational database with three relations, similar to the above schema. Objects can be of any chromosome, indicated by the attribute `chr`.

`s3.res(sts,clone,len,lib,chr);`	A WWW interface, resulting in reports of clones with positive STS, length, library and chromosome.
`s4.seq(sts,primer)`	A CORBA server giving STS names for primer sequences on the X chromosome.

Tab.2 : Available data sources and their relational export schema (left column).

3. Query Correspondence Assertions

To translate a user queries into one or more source queries, we need to have a semantic mapping between the schema of the mediator and the visible schemas of the sources, i.e. the export schemas of the corresponding wrappers. These mapping must be determined by a human operator, how formulates them by defining a set of *query correspondence assertions*. A QCA has the following form:

```
mediator query =: mediator view ⊇ source view := source query
```

and consists of two parts: one describes a query against the mediator (left from the '⊇') and one describes a query against one source (right from the '⊇'). A *mediator query* (MQ) is a conjunctive relational query [1] with arithmetic comparisons against the mediator schema and a *source query* (SQ) is a relational query against the export schema of the wrapper which can be computed by the wrapper; note that the wrapper does not necessarily be able to compute all possible queries, but only those that are used in a QCA. For the purpose of this paper we restrict source queries to conjunctive queries, although this is generally not necessary (see section 4.2). For both queries, views are defined that project out attributes that are not present in the other part.

We will give several examples to illustrate the flavour of QCA. A more rigid definition can be found in [16]. For instance, the following QCA describes source 3:[2]

---

[2] We use a DATALOG like style for queries.

```
(Q1)  exp(Sts,Clone),cinfo(Clone,Len,Lib) =: v1(Sts,Clone,Len,Lib) ⊇
           s3.v1(Sts,Clone,Len,Lib) := res(Sts,Clone,Len,Lib,Chr),Chr='X';
```

A QCA is a rule about the relationship between the extensions of two views. The actual bond between the two sides of the rule is formed by the attributes in the views. Both views must have the same arity, and the QCA defines that a tuple for the source view is a semantically correct tuple for the mediator view. For safety, all variables appearing in the views must also appear in the corresponding queries. The reverse does not hold: both queries can contained variables that do not appear in the head. For instance, the attribute 'Chr' in (Q1) is necessary in the SQ to restrict the results to only X chromosome data, but the mediator schema has no equivalent attribute.

On the first view mediators can only execute user queries which directly match the MQ of one or more QCAs. We say that a MQ is *executed* if the corresponding SQ is executed. The resulting tuples of the SV are by definition correct tuples of the MV, and hence form partial tuples for all relations occurring in the MQ. In section 4 we will show how the mediator can use QCAs to answer arbitrary conjunctive queries[3]. QCAs are a powerful way to express semantic differences between relational schema, provided that these differences can be expressed in the form of conjunctive queries[4].

MV (SV) must project out attributes that are not present in SQ (MQ).This is used in the following examples. The MV of Q2 projects out the attribute 'Lib' of the relation 'cinfo' because source 2, respectively this particular view on source 2, does not contain values for this attribute. If Q2 is executed, tuples for 'cinfo' are therefore padded with nulls. The SV of Q3 projects out the attribute 'Int' in a similar fashion.

```
(Q2)  cinfo(Clone,Len,Lib) =: v2(Clone,Len) ⊇
                          s2.v1(Clone,Len) := clone(Clone,Len,Chr),Chr='X';

(Q3)  exp(Sts,Clone) =: v3(Sts,Clone) ⊇
                                      s2.v2(Sts,Clone) := exp(Sts,Clone,Int);

(Q4)  cinfo(Clone,Len,Lib),Len<150,exp(Sts,Clone) =: v4(Sts,Clone,Lib) ⊇
                       s1.v2(Sts,Clone,Lib) := pos(Sts,Clone),ci(Clone,Lib);
```

Note that attributes that are projected out in the views can still be used in the queries, and often they are necessary to achieve the semantic equivalence. For instance, Q4 poses a restriction on clones from source 1 by expressing that all clones from this source will have a length smaller than 150 KB. This information is later used by the planner to avoid shipping queries to source 1 that have a contradicting condition. Note that such conditions have different effects on the different sides of a QCA: 'forgetting' a necessary condition in the mediator query will lead to unnecessary queries being executed, while 'forgetting' a condition in the source query will lead to erroneous data: if for instance Q2 would not restrict data from source 2 to the X chromosome, data from other chromosomes would appear at the mediator level. In this case, the mediator would not have a chance to correct this, as the mediator schema does not have the chromosome attribute.

In general, sources will overlap in the types of data they store, which will be reflected by an overlap in the relations used in the describing MQs. In this case, the best possible

---

[3] An exception is described in section 4.1.

[4] Which is not always the case. For instance, if one source models the profession of a person as two attributes of a relation 'person' (no person is allowed to have more than two professions), while the mediator schema uses an additional relation 'profession' for this purpose, than these differences can not be expressed by a QCA in the present form.

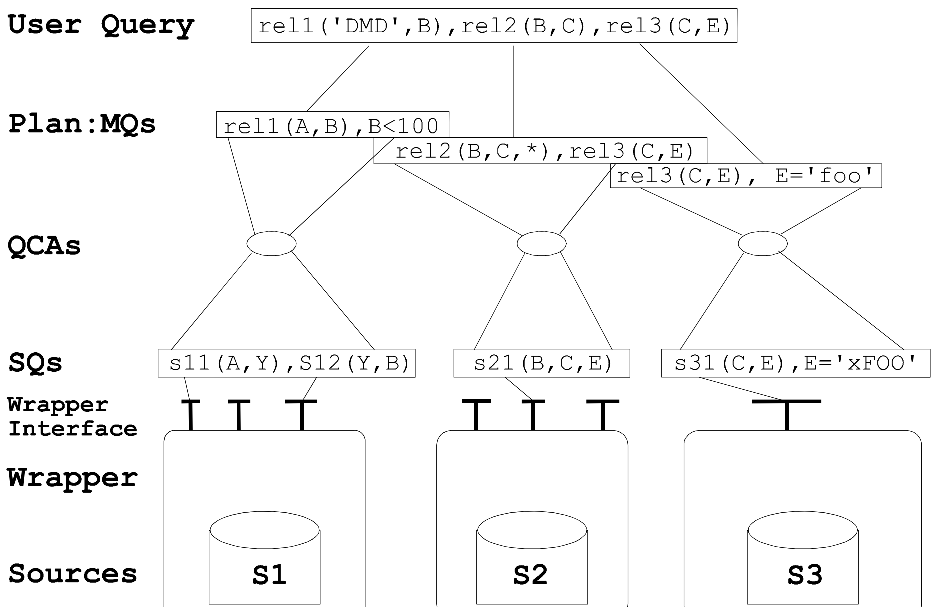

Fig. 3: A query plan is a conjunction of mediator queries, each defined by a source query.

answer at the mediator level is the union of all results of the different wrappers[5]. For instance, lengths of clones are stored in source 2 and 3; considering an individual clone, it is however not known whether both sources. only one or none have a proper value. If both store such a value, these might not coincide. We conclude that both sources must be queried.

4. Query Execution

In this section we describe how a mediator uses a set of QCAs to find an exhaustive answer to a given user query. Exhaustive in this context means that the mediator will try to collect all relevant pieces of data of all sources, as defined through QCAs[6].

4.1 Creating Query Plans using QCAs

Our query planning algorithm is an extension to the algorithm given in [20]. The basic idea draws an analogy to the problem of answering a query using only views [19]. In our setting, the views are the MQs, as they are the queries that the mediator can evaluate by using their SQ counterpart. Given a user query, the problem is to find a combination of these views (a *query plan*, see Fig. 3) that gives an answer to the query. As shown in [19], this is effectively computable for conjunctive queries and conjunctive views with conditions on attribute values. The problem is NP-complete due to the fact that the view conjunctions can not be generated incremental. Consider the following example:

[5] This is the intention behind using the symbol '$\supseteq$' to relate the MQ and the SQ, defining the data from one source as a subset of the data which is intended to appear on the mediator level.
[6] The mediator does not try to find new sources autonomously !

```
User query (UQ1):       r1(X,Y),r2(Y,A),r3(Y,B);
(Q5)  r1(X,Y),r2(Y,A) =: v1(X,Y,A) ⊇ s.v1(X,Y,A) := r(X,Y,A);
(Q6)  r1(X,Y),r3(Y,B) =: v2(X,Y,B) ⊇ s.v2(X,Y,B) := r(X,Y,B);
```

Incrementally finding a plan would proceed as follows: first, try to use Q5 for UQ1 and replace the matching parts, which leads to the query ‘v1(X,Y,A),r3(Y,B)’. But then Q6 is not applicable any more. The same happens if Q6 is be used first. However, considering Q5 and Q6 simultaneously results in a correct plan. It is therefore necessary to consider all possible combinations of source views. Fortunately, [19] shows that it is sufficient to generate only plans of QCAs which contain at most as many QCAs as the user query has predicates.

The planning algorithm proceeds as follows (details can be fond in [16]): it first enumerate all query plans, ignoring those that do not at least contain all relation of the user query. For each of these plans it tries to prove semantic containment (see [2; 1]). If the prove succeeds, executing the plan will yield correct results for the user query. In the worst case an exponential number of plans has to be checked. However, this is restricted to pathological cases. It only takes place if all sources store semantically highly overlapping data. Furthermore the complexity is dramatically reduced if the user query does not contain any relation more than once[7].

An important question is whether *partial* results are desired or not. We only stipulated that the plan must be *semantically contained* in the user query, not semantically equivalent[8]. Consider the following queries:

```
(UQ2)       cinfo(C,Len,Lib), Len<300
(UQ3)       cinfo(C,Len,Lib), Len<50
```

It is reasonable to use Q4 for UQ2, because the results will fulfil the length condition. Nevertheless Q4 and UQ2 are not equivalent, because UQ2 is not contained in Q4. Regarding UQ3, Q4 should be ignored: there is no guarantee that answers are correct, and the mediator is not able to enforce the condition later, because the source does not export a ‘length’ attribute. Considering plans that are not equivalent, but semantically contained, of course greatly increases the number of correct plans.

Note that technically not all user queries are answerable, but only those for which appropriate combinations of source queries exist. If such a combination exists, the planner will certainly find it; but whether or not this is the case is not easy to see. One easy-to-see condition is that each mediator relation appears in at least one MQ; if not, queries involving this relation are not answerable.

4.2 Executing Query Plans

Using the algorithm described above, each user query will result in a number of query plans, if such plans exist. In this section we describe how we execute complete plans.

We first consider a single plan. Remember that each plan consists of a number of MQs. These MQs are connected through inter-QCA joins which are generated by the planning algorithm by renaming variables where necessary. Each MQ is directly associated with a SQ through a QCA, addressing exactly one wrapper. Each SQ is a query against the export schema of the wrapper which is directly executable. We execute SQs sequentially in the order they appear in the plan.

A naiv approach proceeds as following: each wrapper is only considered to be able to execute queries as they are defined in the SQ parts of its QCAs. The result is always the

[7] For such *acyclic* queries only polynomial time is necessary.
[8] which could be tested by proving containment in both directions.

complete extension of the corresponding SV. Enforcing other conditions on attribute values or joining multiple queries addressing the same wrapper is performed by the mediator.

This approach has minimal requirements on the sources, but is certainly not optimal in cases where sources are more powerful. Most sources are at least capable of enforcing equality conditions on attribute values; however, it is not possible (and not reasonable) to define one QCA for each combination of values for all attributes. By using the query capabilities offered by a source, the amount of data to be transferred between the wrapper and the mediator can be drastically reduced. In case the source being a full-fledged relation databases, the source query can be enriched with all conditions of the user query and then be pushed completely.

We therefore allow each wrapper to specify its *query capabilities*:

- For each attribute of a SQ, the wrapper defines the set of predicates it can enforce on them (equality, arithmetic comparisons, string matching etc.).
- The wrapper defines whether it can execute joins between different SQs.

To minimise the number of remote requests and distribute the load of processing, the mediator will try to push as much as possible of the processing to the wrappers. For each SQ the mediator uses the capability descriptions to decide which parts can be pushed to the wrapper. The resulting values are stored in a mediator-specific RDBMS which mimics the export schemas of all wrappers. To ensure also those conditions that could not be pushed, the SQ is evaluated on these tables. The resulting tuples are used to populate the relations of the mediator schema as prescribed by the MQs. This is repeated for every SQ of the query plan, resulting in a set of values in the local RDBMS that are correct answers to the query.

These steps are carried out for every plan, resulting in a comprehensive collection of values in the local RDBMS. Merging results, as for instance described in [22] could be applied now. The original user query is finally executed directly on the mediator schema.

4.3 Query Optimisation

In our experience, the delay caused by network traffic by far dominates the time needed for local query processing or planning. The general goal of query optimisation in our environment is therefore:

- to minimise the number of requests to data sources and
- to minimise the amount of data being transferred through the network.

Query optimisation is on open research area in our project. We give some preliminary ideas here.

Using complete sources: Sometimes there exist one source that by default contains all values for a certain concept. Imagine source 4 would be an absolutely reliable database storing all STSs and their primer sequences. We express this by changing the symbol in the QCA from '⊇' to equality ('='):

```
(Q7)      sinfo(Sts,Primer) =: v5(Sts,Primer) =
                                   s4.v1(Sts,Primer) := s4.seq(Sts,Primer);
```

The idea is that now only this source will be considered for the 'Primer' values, which reduces the number of plans.

Multi-query optimisation: The same SQ is accessed multiple times during the execution of a user query if it appears in multiple plans. This can lead to unnecessary network traffic. Consider the following query plans, obtaining positive clones for an STS with a certain primer:

```
(P1)     s4.v1(Sts,Primer),Primer='ACCTAG',s1.v2(Clone,Sts,Lib);
(P2)     s4.v1(Sts,Primer),Primer='ACCTAG',s2.v2(Sts,Clone);
(P3)     s4.v1(Sts,Primer),Primer='ACCTAG',s3.v1(Sts,Clone,Len,Lib);
```

In this case it would suffice to access source 4 only once.

Removing redundant plans: If a source offers different queries, these are sometimes overlapping. For instance, typical WWW interfaces in molecular biology offer different options for different users. Mostly one can find a simply search form, allowing to search objects by name, and a more sophisticated form with entries for specifying complex conditions. We also observed that the former query form is often very quick, because the underlying database is optimised for such queries, while the latter needs considerable more time. It is therefore reasonable to define QCAs for both cases - but this will lead to redundant plans if both query types are applicable. Such redundancies can be removed by searching for plans that semantically contain other plans. Another question raised in this context is: which of two equivalent plans should be used ?

Bulk queries: Consider the case that a plan generates multiple values for a variable that is used to join another relation in a subsequent plan. In our current approach, that will lead to as many requests to the appropriate view that contains the second relation as there are values resulting from the first view, which is possible the only way to do this for most sources. But some sources could handle that in one query. For instance, a relational database could translate such cases by using a set-construct such as 'where X in (...)'.

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 ([21; 20]). Our approach is an extension of this projects in that it uses a more flexible way to model data sources and considers query optimisation techniques within one unified framework. However, our project does for instance not handle binding patterns, an important restriction of many data sources (see [24]). Fully regarding binding patterns removes the upper bound on the length of plans. [6] describes a method to generate recursive plans in such cases, and also treats certain types of functional dependencies. [28] summarises the theoretic foundations of semantic query containment and discusses also the use if negation. In [14], a similar approach is used, but with a different semantic of queries, leading to different algorithms.

[11] gives an excellent overview over ongoing research in this area, and observed that many projects in information integration can be classified as either '*global-as-view*' or '*local-as-view*', meaning that either the global relations are defined as views over the source schemas (e.g. [27]) or the local relations are defined as views over the global schema (e.g. [20]), respectively. Comparing this with QCAs, the former comes down to restrict MQs to single relations, while the latter is equivalent to restricting SQs to single relations. We allow complex queries on both sides, which results in a higher flexibility. For instance, many examples in section 3 could not be expressed with either of those restrictions. One could argue that it is possible to use the local-as-view strategy by simply defining each SQ as a view on

the source which is implicitly translated inside the wrapper. But this would require mediator-specific wrappers, increasing the effort for data sources to participate in an integrating system. Furthermore, some optimisation possibilities are lost, since the mediator holds less knowledge about the sources.

Our way to describe query capabilities was inspired by the Garlic project ([8]). They push entire queries to the wrappers and let each wrapper decide individually which part it can execute itself and which has to be handled by the mediator. They give examples where this is more efficient than an approach like ours, where only the mediator decides what operations are pushed and which not. However, it requires a back-and-forth sending of plans between mediators and wrappers before actually performing any work. If wrappers reside on the same hosts as the data sources, this will add network requests in first place. Garlic does not consider any semantic conflicts between mediator and sources.

[7] uses a notion similar to QCAs on the first glance. However, their query evaluation procedure can only treat single classes on the mediator side, which comes down to a global-as-view approach. An adventurous feature they include is the use of a multidatabase query language for the SQ parts. This gives better possibilities to merge results from different sources, defining the merging rules on a class-by-class basis. The disadvantage is that deleting a source now requires a careful revision of all rules.

6. Discussion and Future Directions

We presented a new approach to the problem of integrating heterogeneous, distributed data sources with diverging query capabilities. The approach is based on the usage of query correspondence assertions, which offer an elegant and declarative mechanism to express semantic differences in relational schema. Using QCAs, the wrapper and mediator development can be de-coupled to a large degree, which greatly enhances the overall maintainability of the system (see [17] for details). Adding or deleting sources does not affect the mediator. Changes in the schema of a source requires only an adjustment of a set of rules, completely independent from any other source. QCAs are directly used for query execution, where they define the rules after which source views are materialised in a local databases.

It is an important feature that diverging values for the same piece of data are naturally captured, as the approach does not enforce any integrity on data sources. This might appear rather as a bug than a feature for other domains, but is essential in the field of molecular biology.

Query optimisation remains an important and active area for further research. In section 4.3 we gave some ideas in which direction this research will be heading. A detailed analysis will yield many other interesting questions.

Acknowledgements

This work was supported by the German Research Society grant GRK 316.

References

[1] Abiteboul, S., R. Hull, et al. (1995). Foundations of Databases. Reading, Massachusetts, Addison-Wesley Publishing Company.

[2] Aho, A. V., Y. Sagiv, et al. (1979). “Efficient Optimization of a Class of Relational Expressions.” ACM Transactions on Database Systems **4**(4): 435-454.

[3] Arens, Y., C.-N. Hsu, et al. (1996). Query Processing in the SIMS Information Mediator. Advanced Planning Technology. A. Tate. Menlo Park, California, AAAI Press**:** 61-69.

[4] Ashish, N. and C. A. Knoblock (1997). Wrapper Generation for Semi-Structured Internet Sources. ACM SIGMOD Workshop on Management of Semistructured Data, Tuscon, Arizona pp. .

[5] Atzeni, P., G. Mecca, et al. (1997). Semistructured and Structured Data in the Web: going back and forth. ACM SIGMOD Workshop on Management of Semistructured Data, Tuscon, Arizona pp. .

[6] Duschka, O. M. and A. Y. Levy (1997). Recursive Plans for Information Gathering. 15th International Joint Conference on Artificial Intelligence, Nagoya, Japan pp. .

[7] Florescu, D., L. Raschid, et al. (1995). Using Heterogenous equivalence for query rewriting in multidatabase systems. 3rd Int. Conf. on Cooperative Information Systems, Vienna, Austria pp. .

[8] Haas, L. M., D. Kossman, et al. (1997). Optimizing Queries across Diverse Data Sources. 23rd Conference on Very Large Database Systems, Athens, Greece pp. .

[9] Hammer, J., H. Garcia-Molina, et al. (1997). Extracting Semistructured Information from the Web. ACM SIGMOD Workshop on Management of Semistructured Data, Tuscon, Arizona pp. .

[10] Hammer, J. and D. McLeod (1993). “An Approach to Resolving Semantic Heterogeneity in a Federation of Autonomous, Heterogeneous Database Systems.” Journal for Intelligent and Cooperative Information Systems **2**(1): 51-83.

[11] Hull, R. (1997). Managing Semantic Heterogeneity in Databases: A Theoretical Perspective - Invited Tutorial. 16th ACM Symposium on Principles of Database Systems pp. .

[12] InfoBiogen (1998). DBCat - The Public Catalog of Databases.Infobiogen, France . WWW Page . http://www.infobiogen.fr/services/dbcat.

[13] Kim, W., I. Choi, et al. (1995). On Resolving Schematic Heterogeneity in Multidatabase Systems. Modern Database Systems. W. Kim. New York, ACM Press, Addison-Wesley Publishing Company**:** 521-550.

[14] Kwok, C. T. and D. S. Weld (1996). Planning to Gather Information.University of Washington, Department of Computer Science & Engineering . Technical Report . UW-CSE-96-01-04.

[15] Lee, A. J., E. A. Rundensteiner, et al. (1993). An Information Model for Genome Map Representation and Assembly. 2nd International Conference on Information and Knowledge Management, New York, ACM Press pp. 75-84.

[16] Leser, U. (1998). Combining Heterogeneous Data Sources through Query Correspondence Assertions. 1st Workshop on Web Information and Data Management, in conjunction with CIKM'98, Washington, D.C. to appear.

[17] Leser, U. (1998). Maintenance and Mediation in Federated Databases. 8th Workshop on Information Technology and Systems, Helsinki, Finland, to appear.

[18] Leser, U., H. Lehrach, et al. (1998). “Issues in Developing Integrated Genomic Databases and Application to the Human X Chromosome.” Bioinformatics **14**(7): 583-690.

[19] Levy, A. Y., A. O. Mendelzon, et al. (1995). Answering Queries using Views. 14th ACM Symposium on Principles of Database Systems, San Jose, CA pp. 95-104.

[20] Levy, A. Y., A. Rajaraman, et al. (1996). Querying Heterogeneous Information Sources using Source Descriptions. 22th Conference on Very Large Databases, Bombay, India pp. 251-262.

[21] Levy, A. Y., D. Srivastava, et al. (1995). “Data model and query evaluation in global information systems.” Journal of Intelligent Information Systems - Special Issue on Networked Information Discovery and Retrieval **5**(2): 121-143.

[22] Papakonstantinou, Y., S. Abiteboul, et al. (1996). Object Fusion in Mediator Systems. 22th Conference on Very Large Data Bases, Bombay, India, Morgan Kaufmann pp. .

[23] Pitoura, E., O. Bukhres, et al. (1995). “Object Orientation in Multidatabase Systems.” ACM Computing Survey **27**(2): 141-195.

[24] Rajaraman, A., Y. Sagiv, et al. (1995). Answering Queries using templates with binding patterns. 14th ACM Symposium on Principles of Database Systems, San Jose, CA, ACM Press pp. 105-112.

[25] Reck, C. and B. Koenig-Ries (1997). An Architecture for Transparent Access to Semantically Heterogeneous Information Sources. 1st Int. Workshop on Cooperative Information Agents - LNCS 1202, Kiel, Germany pp. 260 -271.

[26] Sheth, A. and J. A. Larson (1990). “Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases.” ACM Computing Survey **22**(3): 183-236.

[27] Tomasic, A., L. Raschid, et al. (1996). Scaling Heterogeneous Databases and the Design of DISCO. 16th Int. Conference on Distributed Computing Systems, Hong Kong, IEEE Computer Society pp. 449-457.

[28] Ullman, J. D. (1997). Information Integration using Logical Views. 6th Int. Conference on Database Theory; LNCS 1186, Delphi, Greece, LNCS 1186, Springer pp. 19-40.

[29] Wiederhold, G. (1992). “Mediators in the Architecture of Future Information Systems.” IEEE Computer **25**(3): 38-49.