Capability Based Mediation in TSIMMIS

Chen Li
Stanford University
chenli@cs.stanford.edu

Ramana Yerneni
Stanford University
yerneni@cs.stanford.edu

Vasilis Vassalos
Stanford University
vassalos@cs.stanford.edu

Hector Garcia-Molina
Stanford University
hector@cs.stanford.edu

Yannis Papakonstantinou
University of California, San Diego
yannis@cs.ucsd.edu

Jeffrey Ullman
Stanford University
ullman@cs.stanford.edu

Murty Valiveti
Stanford University
murty@db.stanford.edu

1 Introduction

The TSIMMIS system [1] integrates data from multiple heterogeneous sources and provides users with seamless integrated views of the data. It translates a user query on the integrated views into a set of source queries and post-processing steps that compute the answer to the user query from the results of the source queries. TSIMMIS uses a *mediation* architecture [11] to accomplish this (Figure 1).

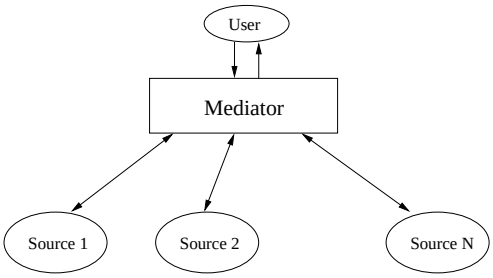

Figure 1: The TSIMMIS Architecture

Many other data integration systems like Garlic [2, 9] and Information Manifold [4] employ a similar architecture. One of the distinguishing features of TSIMMIS is its use of a semi-structured data model (called the *Object Exchange Model* or *OEM* [7]) for dealing with the heterogeneity of the data sources. In particular, it employs source *wrappers* [3] that provide a uniform OEM interface to the mediator.

In SIGMOD-97, we demonstrated a wrapper toolkit in TSIMMIS that helps in efficient development of wrappers based on declarative specifications [3]. We will show how the same specifications are used by the TSIMMIS mediator to generate feasible query plans in the presence of limited source capabilities.

1.1 Problem Definition

Conventional mediators focus on the contents of the sources and their relationship to the integrated views provided to the users. They do not keep track of the capabilities of sources to answer queries. This may lead them to generate plans involving source queries that cannot be answered by the sources. We illustrate this by the following example.

Suppose we have two sources s_1 and s_2 that supply information about conference papers. Let source s_1 have the `title`, `author` and `abstract` of the papers and let source s_2 store `conference` and `title` information. The system provides the integrated view *paper*, which combines the information from the two sources to provide `title`, `author`, `abstract` and `conference` information for each paper. Suppose the user wants to know the titles and abstracts of all the papers written by 'Smith' in 'SIGMOD-97'. The mediator may process this user query by sending a source query to s_1 to get the titles and abstracts of papers written by 'Smith', and sending a query to s_2 to get the titles of all papers in 'SIGMOD-97'. The mediator may then join the results of these two source queries to find the answer to the user query. Suppose s_1 can only return the information about papers, given their titles. Then the source query sent by the mediator to s_1 will fail. Consequently, the mediator will be unable to answer the user query.

1.2 The TSIMMIS Solution

In the above example, the TSIMMIS mediator notices the limited query capabilities of s_1 and finds a feasible plan that respects these limitations. In particular, it generates the feasible plan that first sends a source query to s_2 to find all the titles of papers in 'SIGMOD-97'. For each of these titles, the plan then sends a separate source query to s_1 to verify that the author is 'Smith' and to get the abstract.

2 The TSIMMIS Mediator and Source Capabilities

In this section, we describe how the TSIMMIS mediator processes user queries. In particular, we show how the mediator translates a user query into a set of relevant source queries in Sections 2.1 and 2.2. We also explain in Section 2.3 how the source capabilities can be described by using query templates.

2.1 Query Translation

The mediator encodes the relationship between the user views and the source views with a set of view definitions. Specifically, it uses the *Mediator Specification Language* (MSL) [6] to define user views. MSL is a logic-based language with object-oriented features. For example, the user view *paper* is defined as follows:

```
<paper {<title T><author A><abs B><conf C>}> :-
       <entry {<title T><author A><abs B>}>@s1,
       <entry {<title T><conf C>}>@s2
```

The above rule states that *paper* is essentially a join of the views exported by s_1 and s_2, with `title` being the join attribute. Each rule in MSL has a head and a tail that consist of OEM objects. OEM is a self-describing data model in which each object has a `label` field and a `value` field. The value field can be atomic or complex. Complex values accommodate object nesting. For example, in the MSL rule describing the *paper* view, the head consists of an OEM object with label `paper` and a list of sub-objects describing the `title`, `author`, `abstract`, and `conference` of the `paper` object. Uppercase identifiers (e.g., `T, A`) are variables that are bound to specific values at run time.

Suppose the user wants to find the `title` and `abstract` of each paper written by 'Smith' in 'SIGMOD-97'. The user formulates the following query, based on the user view *paper*:

```
<ans {<title T><abs B>}> :-
     <paper {<title T><author 'Smith'><abs B>
            <conf 'SIGMOD-97'>}>
```

When the user query arrives at the mediator, the mediator uses the view definitions to translate the query on the user views into a *logical plan* [6] (i.e., a set of MSL rules which refer to the source views instead of the integrated views). The following is the logical plan for the example user query:

```
<ans {<title T><abs B>}> :-
     <entry {<title T><author 'Smith'><abs B>}>@s1,
     <entry {<title T><conf 'SIGMOD-97'>}>@s2
```

We refer to the parts of the right hand sides of the logical query plan rules as *conditions*. In the logical plan above, there is only one rule with two conditions (there will be multiple rules in the logical plan only if there are multiple rules for the user view *paper*).

2.2 Physical Plans

The logical query plans in TSIMMIS do not specify the order in which the conditions are processed. This is done in the physical plans generated in the subsequent stages of the TSIMMIS mediator.

Three possible physical plans for the logical plan of the example user query are:

- P_1: Send query `<entry {<title T><author 'Smith'> <abs B>}>` to s_1 [1]; send query `<entry {<title T> <conf 'SIGMOD-97'>}>` to s_2; join the results of these source queries on the `title` attribute.
- P_2: Send query `<entry {<title T><author 'Smith'> <abs B>}>` to s_1; for each returned `title`, send query `<entry {<title T> <conf 'SIGMOD-97'>}>` to s_2, with `T` bound.
- P_3: Send `<entry {<title T><conf 'SIGMOD-97'>}>` to s_2; for each returned `title`, send `<entry {<title T><author 'Smith'><abs B>}>` to s_1, with `T` bound.

Some of these physical plans may or may not be feasible depending on the query capabilities of the sources, as discussed in Section 2.3.

2.3 Source Capabilities Description: Templates

In order to describe the capabilities of sources, the TSIMMIS system uses templates to represent sets of queries that can be processed by each source [3, 5]. Suppose s_1 and s_2 have the following templates.

T_{11}: `X:-X:<entry {<title $T><author A><abs B>}>@`s_1
T_{21}: `X:-X:<entry {<title T><conf $C>}>@`s_2
T_{22}: `X:-X:<entry {<title $T><conf C>}>@`s_2

The first template T_{11} says that source s_1 can return all the information it has about a paper given its `title`. T_{21} says that s_2 can return all the information it has about papers given the `conference`. T_{22} says that s_2 can also return the information about a paper given its `title`. Assume that these are the only templates for s_1 and s_2. That is, s_1 and s_2 cannot answer any other kinds of queries.

Given these capabilities, P_1 (Section 2.2) is not feasible since s_1 cannot answer the query `<entry {<title T><author 'Smith'><abs B>}>`. This is because the `title` value is not specified. P_2 is also infeasible for the same reason. Only P_3 is feasible as the mediator first gets the `title` of each paper in 'SIGMOD-97' from s_2 and uses this `title` value to get the corresponding `abstract` information from s_1 and check that the author is indeed 'Smith'. Notice that the queries to s_1 are now feasible because they specify the `title` values.

Other ways to describe source capabilities in mediator systems have been proposed in the literature [4, 8]. For instance, the Information Manifold [4] uses *capabilities records* for encoding source capabilities. We note that the capability description language used by TSIMMIS is more powerful than the ones proposed in [4] and [8], as shown in [10].

In the next section, we will show how the plan generation process in TSIMMIS takes into account the source capabilities in producing feasible query plans.

3 Capability Based Plan Generation

The architecture of TSIMMIS mediator is shown in Figure 2(a). The logical plan generated by the view expander is passed onto the plan generator module, which computes a feasible physical plan for the query. As shown in Figure 2(b), it accomplishes this in three stages.

3.1 Matcher

The first step in the plan generation process is to find all the source queries that can process parts of the logical plan. Some of these queries have requirements indicating the list of variables that need to be bound. To illustrate, consider the logical plan of Section 2.1. Let the two conditions of the logical plan be denoted C_1 and C_2. There is one query to process C_1, with the requirement that variable `T` be bound.

[1] Strictly speaking, source queries are MSL rules. For simplicity, we just show the tail of the source query rule.

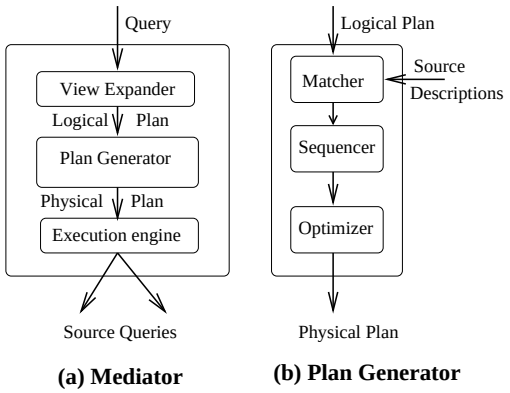

Figure 2: Mediator architecture

Source Query	Source Template	Condition Processed	Binding Requirement
M_1	T_{11}	C_1	T
M_2	T_{21}	C_2	None
M_3	T_{22}	C_2	T

Table 1: Matcher Result

There are two queries that process C_2: one with T bound and another without any binding requirements.

Table 1 describes the result of the Matcher. It has a row for each source query that processes some conditions of the logical plan. For instance, the first row M_1 in the table indicates that s_1 can process C_1 by using template T_{11} with the requirement that variable T be bound.

3.2 Sequencer

The second step in the plan generation process is to piece together the source queries for processing the conditions of the logical plan in order to construct feasible plans. Here, what matters is not just the specific source queries chosen to cover all the conditions of the logical plan but also the sequence of processing these queries.

The Sequencer finds the set of feasible sequences of source queries. Each query in a feasible sequence has the property that the variables in its binding requirements are exported by the source queries that appear earlier in the sequence. For instance, in our example logical plan, the Sequencer finds that the only feasible sequence is $< M_2,\ M_1 >$, with source query M_1 being parameterized (variables bound) from the result of the source query M_2. Sequence $< M_1,\ M_2 >$, though it can also process all the conditions, is not feasible because M_1's binding requirements cannot be satisfied. Other sequences like $< M_2,\ M_3 >$ cannot process all the conditions of the logical plan.

3.3 Optimizer

Having found the feasible sequences of source queries, the third step of the plan generation process is to optimize over the set of corresponding feasible plans and choose the most efficient among these. The Optimizer uses standard optimization techniques to pick the best feasible plan and translates it into a physical plan. In our example case, there is only one feasible sequence of queries $< M_2,\ M_1 >$ and this leads to the physical plan P_3 of Section 2.2.

4 Implementation and Demonstration

We have implemented capability based plan generation in the TSIMMIS mediator as described above. In the demonstration, we show how the mediator constructs feasible plans for user queries in the presence of limited source capabilities.

We employ four information sources in the demonstration. These sources come from two different domains: bibliographic data and financial information. The financial information comes from web-based sources and the bibliographic data comes from legacy systems.

We encode the capabilities of the sources by specifying a set of templates for each source. We demonstrate the ability of the TSIMMIS mediator to generate plans for user queries by taking into consideration these source capabilities. The demonstration is run through a web-based interface to the TSIMMIS mediators.

References

[1] H. Garcia-Molina et al. The TSIMMIS approach to mediation: data models and languages. *Journal of Intelligent Information Systems*, vol. 8, pages 117-132.

[2] L. Haas, D. Kossman, E. Wimmers, and J. Yang. Optimizing Queries Across Diverse Data Sources. In *Proc. VLDB Conference*, 1997.

[3] J. Hammer, H. Garcia-Molina, S. Nestorov, R. Yerneni, M. Breunig and V. Vassalos. Template-Based Wrappers in the TSIMMIS System. In *Proc. ACM SIGMOD Conference*, 1996.

[4] A. Levy, A. Rajaraman and J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *Proc. VLDB Conference*, 1996.

[5] Y. Papakonstantinou, A. Gupta, and L. Haas. Capabilities-based Query Rewriting in Mediator Systems. In *Proc. PDIS Conference*, 1996.

[6] Y. Papakonstantinou, H. Garcia-Molina, J. Ullman. Medmaker: A Mediation System Based on Declarative Specifications. In *International Conference on Data Engineering*, 1996.

[7] Y. Papakonstantinou, H. Garcia-Molina, J. Widom. Object Exchange Across Heterogeneous Information Sources. In *Proc. ICDE Conference*, 1995.

[8] A. Rajaraman, Y. Sagiv and J. Ullman. Answering Queries Using Templates with Binding Patterns. In *Proc. ACM PODS Conference*, 1995.

[9] M. Tork Roth, and P. Schwarz. Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Sources. In *Proc. VLDB Conference*, 1997.

[10] V. Vassalos and Y. Papakonstantinou. Expressive Capabilities Description Languages and Query Rewriting Algorithms. Stanford Technical Report, 1997.

[11] G. Wiederhold. Mediators in the Architecture of Future Information Systems. In *IEEE Computer*, 25:38-49, 1992.