Capability Based Mediation in TSIMMIS

Chen Li, Ramana Yerneni, Vasilis Vassalos
Hector Garcia-Molina,Yannis Papakonstantinou, Jeffrey Ullman

Department of Computer Science
Stanford University
Stanford, CA 94305, USA
{chenli, yerneni, vassalos, hector, yannis, ullman}@cs.stanford.edu

1 Introduction

The TSIMMIS system [5] 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 to compute the answer to the user query from the results of the source queries. TSIMMIS uses a *mediator* architecture [9] to accomplish this (see Figure 1).

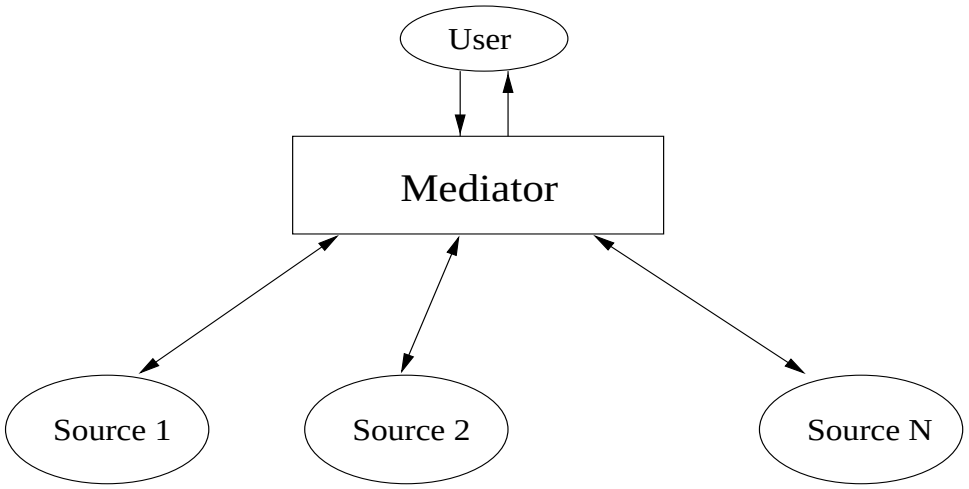

Figure 1: The TSIMMIS System

We illustrate the query processing of TSIMMIS with 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 have $conference$ 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 "John Smith" in "SIGMOD97". 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 "John Smith", and sending a query to s_1 to get the titles of all papers in "SIGMOD97". The mediator may then join the results of these two source queries to find the answer to the user query.

Traditional mediators focus their attention 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. In the above example, the mediator answers the user query by sending a source query to s_1 to get the titles and abstracts of all papers written by "John Smith". 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.

In order to solve this problem, mediators should incorporate information about the capabilities of sources in the process of their plan generation. In the above example, the mediator should notice the limited query capabilities of s_1 and find a feasible plan that respects these limitations. In particular, it should find the feasible plan that first sends a source query to s_2 to find all the titles of papers in "SIGMOD97". For each of these titles, the plan is to send a separate source query to s_1 to verify that the author is "John Smith" and get the abstract.

In TSIMMIS, we describe source capabilities using query templates (see Section 2.3) as discussed in [4] and [2]. Other systems like Information Manifold ([3], [6]) have used source capability descriptions based on binding patterns adorning the views exported by sources. For example, in

the case of s_1's capability limitation described above, the adornment is bff specifying that queries on s_1's view can only be answered if the *title* attribute is bound (the *author* and *abstract* attributes can be either bound or free). We note here that our source description language is strictly more powerful than the one used by Information Manifold. Moreover, TSIMMIS explores the space of feasible plans to find an efficient plan while Information Manifold aims to generate only one feasible plan for the user query.

In this paper, we show how the TSIMMIS mediator takes into account the capabilities of the sources to generate feasible query plans for user queries. Section 2 explains how the mediator processes user queries in general, and also discusses the way in which source capabilities are described in TSIMMIS. Section 3 describes the details of the capability based plan generation process. The demo content is indicated in Section 4.

2 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 the user queries 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) [5] to define user views. The view definitions in MSL are similar to Datalog rules [8]. For example, the user view *paper* is defined as follows:

```
<paper {<title T> <author A> <abstract B> <conference C>}> :-
<entry {<title T> <author A> <abstract B>}>@s1 AND <entry {<title T> <conference 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.

Suppose the user wants to find the *title* and *abstract* of each paper written by "John Smith" in "SIGMOD97". The user formulates the following query, based on the user view *paper*:

```
<ans {<title T> <abstract B>}> :-
<paper {<title T> <author ''John Smith''> <abstract B>
<conference ''SIGMOD97''>}>
```

When the user query arrives at the mediator, the mediator uses the view definitions to translate the query on the user views into a set of queries on the source views. This process of view expansion yields a logical query plan for the user query. The following is the logical plan for the example user query:

```
<answer {<title T> <abstract B>}> :- <entry {<title T> <author ''John Smith''> <abstract
B>}>@s1
AND <entry {<title T> <conference ''SIGMOD97''>}>@s2
```

The logical query plan is a set of MSL rules, each specifying how a set of answers to the user query is computed from a set of source queries. Sometimes we refer to the source queries specified on 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*). The rule states that answers to the user query can be computed by sending two source queries. The first one, to s_1 gets the *title* and *abstract* of each entry (for a paper) corresponding to "John Smith", while the second one, to s_2, gets the *title* of each paper in conference "SIGMOD97". From the results of the two source queries, the bindings for variables T and B are obtained to construct the answers to the user query.

2.2 Physical Plans

The logical query plans in TSIMMIS do not specify the order in which the conditions are processed (i.e., source queries sent to the sources). 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 "$< entry\ \{< title\ T >< author\ \text{“}John\ Smith\text{”} >< abstract\ B >\} >$" to s_1; send query "$< entry\ \{< title\ T >< conference\ \text{“}SIGMOD97\text{”} >\} >$" to s_2; join the results of these source queries on the *title* attribute.

- P_2: Send query "$< entry\ \{< title\ T >< author\ \text{“}John\ Smith\text{”} >< abstract\ B >\} >$" to s_1; for each returned *title*, send parameterized query "$< entry\ \{< title\ T >< conference\ \text{“}SIGMOD97\text{”}\} >$" to s_2 with T bound.

- P_3: Send query "$< entry\ \{< title\ T >< conference\ \text{“}SIGMOD97\text{”} >\} >$" to s_2; for each returned *title*, send parameterized query "$< entry\ \{< title\ T >< author\ \text{“}John\ Smith\text{”} >< abstract\ B >\} >$" to s_1 with T bound.

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 [2]. Suppose s_1 and s_2 have the following templates:

s_1: `X :- X:<entry {<title $T> <author A> <abstract B>}>`

s_2: `X :- X:<entry {<title T> <conference $C>}>`

s_2: `X :- X:<entry {<title $T> <conference C>}>`

The first template says that source s_1 can return all the information it has about each paper given the *title*. The second template says that s_2 can return all the information it has about each paper given the *conference*. The third template says that s_2 can also return all the information 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* "*John Smith*" $>< $ *abstract* $B>\}>$". 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 "SIGMOD97" from s_2 and uses this *title* value to get the corresponding *abstract* information from s_1 and check that the author is indeed "John Smith". Notice that the queries to s_1 are now feasible because they specify the *title* values.

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

Consider the logical query plan and the source templates of Section 2. The task at hand is to generate an efficient feasible physical plan for the logical plan with respect to the source capabilities.

Logical plans in the TSIMMIS mediator indicate the pieces of information that are needed to be obtained from the data sources in order to compute the answers to the user queries. For example, the logical plan of Section 2.1 says that information about *title* and *abstract* of papers written by "John Smith" should be obtained from source s_1 while information about the *title* of papers in "SIGMOD97" should be obtained from source s_2. However, it does not indicate whether or not merely sending the two source queries listed in the logical plan will be acceptable to the sources. In other words, from the logical plan, we do not know if "$<$ *entry* $\{<$ *title* $T ><$ *author* "*John Smith*" $><$ *abstract* $B >\} >$" can be answered by s_1 and if "$<$ *entry* $\{<$ *title* $T ><$ *conference* "*SIGMOD97*" $>\}$" can be answered by s_2.

From the templates describing the capabilities of sources s_1 and s_2, we know that the first condition of the logical plan cannot be processed by directly sending the source query to s_1. There are, however, other ways of evaluating this condition. For instance, if we can bind the variable T, then the source query can be sent to s_1. This leads to the following way to evaluate the logical plan: send the second source query to s_2; for each value of T returned by the query to s_2, send the first source query with T bound by this value to source s_1; use the T and the B values computed from the source queries to construct answers to the user query based on the head of the logical plan rule.

3.1 Plan Generator Module

The above discussion illustrates the main ideas of the capability based plan generation processor. Figure 2 shows the Plan Generator module of the TSIMMIS mediator. This module takes the

logical query plan and the capability description of the sources and computes an efficient feasible physical plan for the user query. It accomplishes this in three stages, as described below.

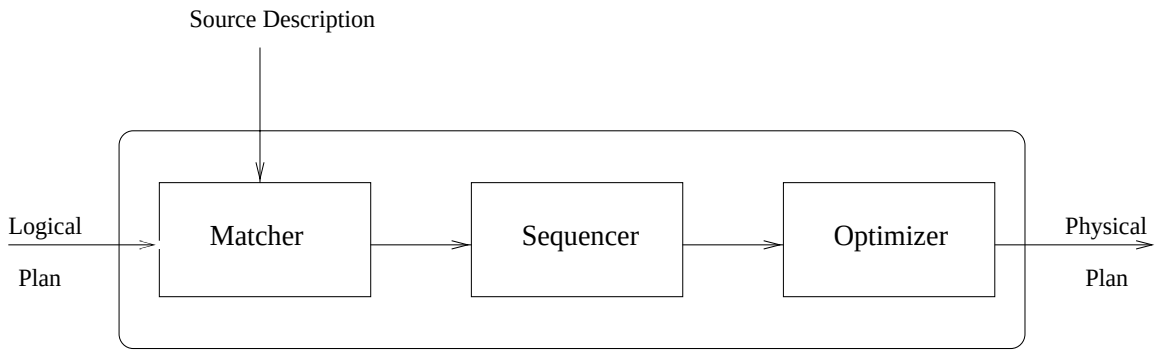

Figure 2: The Plan Generator Module

3.2 Matcher

The first step in the plan generation process is to find all the options for processing each of the source queries specified in the logical plan. Some of these options 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 source queries of the logical plan be denoted C_1 and C_2. There is one option for processing C_1, with the requirement that variable T be bound. There are two options for processing C_2. The first is to send it to s_2 as is and the second is to send it to s_2 by binding T. Denoting the three source templates of Section 2.3 as T_{11}, T_{21} and T_{22}, the following table describes the result of the Matcher:

Condition Processed	Template	Binding Requirement
C_1	T_{11}	T
C_2	T_{21}	None
C_2	T_{22}	T

The Matcher table has a row for each option of processing a condition of the logical plan. For instance, the first row in the above table indicates that C_1 is matched with T_{11} with the requirement that the variable T be bound. That is, C_1 can be sent to source s_1 if T is bound in C_1. The second row of the Matcher table indicates that C_2 can be sent to source s_2 as is because it is matched with the template T_{21} at s_2 without any binding requirements. Finally, another way of processing C_2 at s_2 is to bind T.

3.3 Sequencer

The second step in the plan generation process is to piece together the information about the sets of options for the various source queries of the logical plan in order to construct feasible plans. Here, what matters is not just the specific options chosen for each condition but also the sequence of processing these conditions. For instance, in our example logical plan, there is only one combination of options for the two conditions of the logical plan. Having identified this combination, the mediator also needs to figure out that the proper sequence of processing is the second condition followed by the first. The other sequence, the first condition followed by the second, does not lead to a feasible plan.

The Sequencer uses the table output by the Matcher to find the set of sequences of conditions that constitute feasible plans. Each of these sequences has the property that the binding requirements of a condition in the sequence are satisfied by the results of the source queries of the conditions that appear earlier in the sequence. For instance, using the Matcher table described above, the Sequencer finds that the only feasible sequence is $< C_2, C_1 >$, with source query of C_1 being parameterized (variables bound) from the result of the source query of C_2. The other sequence, $< C_1, C_2 >$ is obviously not feasible because C_1's binding requirements cannot be satisfied in this sequence.

3.4 Optimizer

Having found the feasible sequences of conditions, 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 conditions (i.e., the set output by the Sequencer is a singleton) and this leads to the physical plan P_3 of Section 2.2.

4 Implementation and Demo

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

We employ three sources in the demo:

- Inspec: a university owned legacy system that contains bibliographic information;
- A web source providing information on bibliographic entries;
- A collection of UNIX files containing bibliographic information which is accessible through Perl scripts.

These sources are integrated into the TSIMMIS system by using *wrappers* [2] to provide uniform interfaces to them.

We use a web based graphical query interface called MOBIE [1] in our demo to allow users access to the TSIMMIS system. We encode the capabilities of the sources by specifying sets of templates, one for each source. We demonstrate the ability of the TSIMMIS mediator to generate plans for user queries by taking into consideration these templates. We will also show how changes

to the templates affect the feasible plans generated by the TSIMMIS mediator, by altering the templates online and rerunning the same set of queries.

References

[1] J. Hammer et al. Information Translation, Mediation, and Mosaic-based Browsing in the TSIMMIS System. In *Proc. ACM SIGMOD Conference*, 1995.

[2] 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.

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

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

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

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

[7] J. Ullman. Information Integration Using Logical Views. In *Proc. ICDT Conference*, 1997.

[8] J. Ullman. Principles of Database and Knowledge-Base Systems. Vol. I: Classical Database Systems. *Computer Science Press*, New York, NY, 1988.

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