Coping with Limited Capabilities of Sources

Hector Garcia-Molina[1] and Ramana Yerneni[2]

[1] hector@cs.stanford.edu, Stanford University, USA
[2] yerneni@cs.stanford.edu, Stanford University, USA

Abstract. In various contexts (e.g., the Internet), the query-processing capabilities of data sources may be limited. Middleware systems based on a mediation architecture are employed to provide powerful query processing interfaces to data sources with limited query capabilities. Such systems also provide integrated views to the data across multiple sources. In this paper, we discuss how the query capabilities of data sources can be described and how mediators support extended query sets over the data sources. We also describe how the set of queries supported by a mediator can be computed from the corresponding sets of queries supported by the data sources integrated by the mediator. We present ways in which content description of data sources can help in extending the set of supported queries. Finally, we describe the challenges posed by diverse source capabilities when supporting robust query processing through replicated sources in the presence of source failures.

1 Introduction

In contexts like the Internet, data sources have varying and limited query processing capabilities. Processing queries over such sources and over mediators that provide integrated views over such sources poses many challenges.

Example 1. Consider the Web bookstore Amazon.com [20].[3] Users pose queries to this source by filling out search forms. Amazon.com provides a handful of forms [21, 22, 23], one of which supports search based on author, title, format and subject attributes. Specifically, the user must fill out at least one of the four attributes and submit the form [21]. In response, Amazon.com will return a list of books with some of these and additional attributes. In particular, it does not return the subject attribute and it returns the price attribute among others.

Many simple queries cannot be posed to Amazon.com using this search interface. For instance, suppose the user wants to find the list of books written by "Freud" that cost less than $10. Amazon.com does not support this query or any query that specifies price conditions.

Now, consider another Web bookstore BarnesAndNoble.com [24]. Once again users can pose queries by filling out a search form. BarnesAndNoble.com only supports one search form that allows queries to specify at least one of author, title and keywords attributes, along with additional specification of price, format

[3] All features of real sources mentioned in this paper are as of December 15, 1998.

and other attributes. In particular, BarnesAndNoble.com can support the query that finds all books by "Freud" that cost less than \$10. However, it cannot support many other simple queries like "find all the books that cost less than \$10." While it is true that this query is also not supported by Amazon.com, there are many queries that Amazon.com supports which are not supported by BarnesAndNoble.com. For instance, one such query is "find all books published by McGraw-Hill."

Typically, wrappers that provide access to Web sources are faced with the same limitations human users face. Such wrappers would send HTTP requests to the source, as if the requests came from a browser, and would then extract the data from the returned HTML forms. Thus, the wrapper can only offer the same functionality humans see.

One might wonder why sources have limited query capabilities. There are many reasons for capability limitations. Chief among them are:

1. Sources may choose to provide simple query interfaces through search forms even if they are actually capable of answering more complex queries.
2. Certain attributes may be hidden for security reasons. Such attributes may not appear in the query results and/or they may not appear in the query-condition specification.
3. Sources may only allow conditions on attributes for which they have indexes so that certain query response-time profiles can be maintained.
4. There may be published *application programming interfaces* (APIs) that are in current use, so they continue to support such interfaces even when they are restrictive.

We note that source capabilities tend to change over time. Many sources start with a few restrictive and simple query interfaces. As they build more powerful backend databases and relax security concerns, they tend to provide more complex query capabilities. In a context like the Internet, source capabilities evolve rapidly. Sources tend to change their Web forms in a matter of months if not weeks.

The goal of this paper is to give an overview of the challenges introduced by limited-capability sources, as well as of some of the solutions being explored. To illustrate solutions, we use mainly work done by the Stanford Database Group. We do provide references to other related work, but this paper does not attempt to provide comprehensive coverage.

First, we address the problem of describing source capabilities so that one can reason about capability limitations of sources. In particular, we need to know how to check if a target query is feasible with respect to a given capability description and we also need to understand how the capability descriptions of multiple sources interact when queries spanning multiple sources are processed. In Section 2, we describe a simple framework to express source capabilities in which feasibility checking of target queries is very easy. At the same time the framework is rich enough to accommodate a large variety of capability limitations of data sources, particularly those in the Internet context.

It is important to note that there is a fundamental difference between source capability descriptions (the focus of this paper) and source content descriptions. The former identify which queries are answerable by a source, while the latter define what the source contains. To illustrate the difference, say one wishes to find all books on a given subject at BarnesAndNoble.com. The search form restricts the subject field to be chosen from a menu of choices. If the user wants to find books on "Psychology", BarnesAndNoble.com cannot answer because "Psychology" is not one of the menu choices. However, this does not mean that there are no books on "Psychology" at BarnesAndNoble.com. In fact, one can verify that BarnesAndNoble.com indeed has books on "Psychology" by asking for these books through other search criteria like author or title. Thus, when we construct a capability description for this source, we are only describing what queries are accepted, and we are making no statement about what is actually stored at the source. One could also provide a content description, e.g., stating what subjects exist at the source, but this is orthogonal to the capability description. Our focus in this paper is on capability descriptions, although in Section 6 we explore how content descriptions can help in processing queries over sources with limited capabilities.

As illustrated by Example 1, the limited capabilities of data sources make the processing of even simple queries challenging. In Section 3, we discuss how queries with complex condition specification can be processed on sources with limited capabilities. We describe how mediators use postprocessing techniques to support additional query conditions and how they employ query decomposition techniques to answer queries with complex condition expressions involving disjunctions as well as conjunctions.

In data-integration systems, mediators provide integrated views over many sources that have limited and varying query-processing capabilities. Section 4 describes how mediators cope with the source capability limitations when translating queries on integrated views into source queries that respect the restrictions imposed by the sources. In Section 5, we discuss the need to compute the capabilities of the mediators from those of their sources and describe how mediator capabilities are computed. There we also identify issues surrounding the concise description of mediator capabilities which in turn leads to efficient computation of mediator capabilities and efficient feasibility verification at query-processing time. We also discuss how the interface needs of a mediator can translate to capability requirements on its sources.

Section 6 briefly describes the kinds of source content descriptions that can be useful in increasing the set of user queries for which feasible plans can be found and in executing the plans more efficiently. Then, in Section 7, we discuss how inter-source content descriptions can be used to derive information about alternative sources that can be used in coping with source failures during query processing. In particular, we note a few interesting challenges when alternative sources have unequal query-processing capabilities.

2 Describing Query Capabilities

To illustrate how capabilities can be described, in this section we first present a simple description framework that will be helpful throughout this paper. Then, in Section 2.2, we discuss other options for describing capabilities.

In our framework, each data source exports a set of relational views.[4] Conceptually, a query to a source is submitted by filling out a form on one of the views exported by the source. The query specifies values for some attributes of the view, and the result of executing the query is a set of tuples of the source view on which the query is posed. The following example illustrates a source view and answering queries on that view.

Example 2. Consider a source that exports a view $R(X, Y, Z)$. Let the set of tuples in source view R be $\{(x_1, y_1, z_1), (x_1, y_2, z_1), (x_2, y_2, z_2)\}$. The result of the source query $R(X, Y, z_1)$ is $\{(x_1, y_1, z_1), (x_1, y_2, z_1)\}$, while the result of the source query $R(X, y_1, Z)$ is $\{(x_1, y_1, z_1)\}$. Other sample queries are: $R(x_1, Y, z_1)$, $R(x_1, Y, Z)$ and $R(X, Y, Z)$.

2.1 Attribute Adornments

The query capabilities of a data source are expressed as a set of *templates* supported by the source. A template, like a form on the Web, identifies the various attributes of a source view that can be specified in a query submitted on the view. Restrictions on attribute specification are also indicated by templates (e.g., if an attribute value has to be chosen from a menu of choices). We use attribute *adornments* to specify how the attributes of a view can participate in a query template.

Based on our study of query forms for a variety of Web data sources, we consider the following five kinds of attribute adornments:

1. adornment **f** (for free): the attribute may or may not be specified in the query;
2. adornment **u** (for unspecifiable): the attribute cannot be specified in the query;
3. adornment **b** (for bound): the attribute must be specified in the query;
4. adornment **c**$[S]$ (for constant): the attribute must be specified, and in addition, it must be chosen from the set of constants S;
5. adornment **o**$[S]$ (for optional): the attribute may or may not be specified in the query, and if specified, it must be chosen from the set of constants S.

Each template specifies an adornment for each attribute of a view. In the case of the c and o adornments, the menus of constants allowed are also specified by the templates. The following example illustrates how templates with attribute adornments are used to express query capabilities.

[4] We use the relational framework for simplicity of exposition. We believe that all the main ideas presented in this paper carry over seamlessly to other data models like OEM ([13]) and XML [26].

Example 3. Consider the view $R(X, Y, Z)$ exported by the source of Example 2. Let the source support a set of queries on this view that specify some value for X and no value for Y. In addition, let these queries optionally specify some value for Z. A template that expresses these query capabilities is *buf*. This template is similar to a Web form in which only the X and the Z fields appear, with the annotation that the X field must be specified when submitting the form.

Suppose the source also supports another set of queries in which X cannot be specified at all, Y can be specified optionally while Z must be specified and must be chosen from $\{z_1, z_2\}$. These query capabilities can be expressed by the template *ufc*$[z_1, z_2]$.

Based on the two templates, one can easily determine whether a given query on the source view is answerable or not. For instance, query $R(x_1, Y, Z)$ is answerable because it satisfies the first template, while query $R(X, Y, z_1)$ is answerable because it satisfies the second template. Queries $R(X, y_1, Z)$ and $R(X, Y, z_3)$ do not satisfy either template. So, they are not answerable.

Templates can also be used to describe restrictions on the set of attributes returned when sources answer queries. There are situations in which a view may have attributes on which conditions may be specified, but these attributes are not returned in the answer. For example, we can pose a query to Amazon.com [21] by specifying the subject attribute of the desired set of books. At the same time, Amazon.com does not return the subject attribute when answering queries. For instance, when a user queries Amazon.com for books written by a specified author, the returned list of books does not mention their subject areas. That is, subject is a query attribute, not an output attribute in Amazon.com.

In order to represent sources that do not return certain attributes, we need to specify explicitly the output restrictions of attributes in the templates of the views exported by the sources. In a template, each attribute should be adorned to reflect its input (query) as well as its output (result) restrictions. To describe the input requirements of an attribute that has no output restriction (i.e., it appears in the result), we use the adornments discussed earlier: *f*, *o*, *b*, *c* and *u*. To describe the input restrictions of an attribute whose output is suppressed (i.e., it does not appear in the result), we introduce five new adornments: *f'*, *o'*, *b'*, *c'* and *u'*. To illustrate the use of the new adornments, consider a source that exports view $R(X, Y, Z)$ with the requirement that queries on this view must specify X, must not specify Z, and can specify Y optionally. Let the source suppress Y in its output (i.e., X and Z are output, Y is not). We describe the capabilities of the source with a *bf'u* template on R.

2.2 Other Capability-description Mechanisms

Contemporary systems like TSIMMIS, Information Manifold, DISCO and Garlic are aware of source capability limitations and use a host of capability-description mechanisms. TSIMMIS [2, 5] uses *query templates* to describe source capabilities. The query restrictions are indicated as attribute adornments of *bound*, *free* or bound to a single fixed *constant* (i.e., *b*, *f* and *c* with singleton menus). The capability-description language we discussed in Section 2.1 is more powerful in that it considers a richer set of attribute adornments.

The Information Manifold [7, 8] uses *capability records* to describe source capabilities. Each source has a capability record that indicates the input attributes of the source view, its output attributes, the smallest number of input attributes that must be specified in a query to the source, and the attributes on which the source can apply selections of the form $\alpha \; op \; c$, where c is a constant and $op \in \{\leq, <, \neq, =\}$. DISCO [15, 6] uses a capability-description language that is similar to the capability records of the Information Manifold. Both these description languages cannot express adequately the requirements of menus of choices for attribute specification.

Garlic [1, 4] is a mediation system that integrates large-scale multimedia information systems. The metadata in Garlic does not include the query-processing capabilities of the individual sources, so the query processor at the mediator has no direct knowledge about the capabilities of the sources being integrated. In order to make sure that the plan generated by the mediator is feasible, the sources participate in the mediator's query-planning process. That is, each source encapsulates the knowledge of its own capabilities and lets the mediator know if a given query is answerable by the source.

In [3] we studied a capability-description language based on context-free grammars. In this language, one can express boolean query-condition expressions involving disjunctions as well as conjunctions. The simple language based on templates discussed in Section 2.1 is strictly less powerful than the language presented in [3]. In particular, query templates based on attribute adornments can be expressed in the language of [3] while disjunctive condition expressions represented in the language of [3] cannot be described by query templates. In a way, one can view a query template as a simple conjunctive query-condition expression in which the constituent atomic conditions correspond to the attributes with *b*, *b'*, *c*, *c'*, *f*, *f'*, *o* and *o'* adornments.

3 Extending Source Capabilities

In order to extend the set of supported queries of a source with limited capabilities, middleware systems called *mediators* employ a variety of techniques. Mediators present external agents and end users with more powerful interfaces to data sources. That is, a query to a source that is ordinarily rejected, because it is not in the set of supported queries of the source, may be accepted by a mediator for that source, and processed successfully. Here we discuss two important techniques employed by mediators and show how they extend the query capabilities of a source.

3.1 Postprocessing at the Mediator

Mediators often use the ability to postprocess the results of source queries in order to support query conditions that cannot be "pushed" to the sources. Suppose, a user query on a source view specifies a value for an attribute X that has the u adornment. That is, the source does not allow queries to specify values for X, but it returns the X attribute in its output. When the user query is submitted to a mediator of the source, the mediator can send an answerable query to the source that does not specify the X value. On receiving the results of the source query, the mediator can filter the results using the X condition in the user query. If a query that is directly supported by a source is submitted to the mediator, it can pass on the query to the source without any modification and transfer the results of the source query to the user without any postprocessing.

Example 4. Consider the Internet bookstore Amazon.com. In its query interface, Amazon.com does not allow the specification of price conditions. For instance, the query ($author$ = "$Sigmund\ Freud$" $\wedge$ $price < \$10$) is not supported by Amazon.com. However, a mediator running on top of Amazon.com can accept this query, send the feasible query ($author$ = "$Sigmund\ Freud$") to Amazon.com and filter its results with the condition ($price < \$10$). Thus, the mediator can extend the set of queries supported by Amazon.com by performing the necessary postprocessing on the results of the source queries.

Most contemporary mediator systems (e.g., TSIMMIS [5], Information Manifold [8], DISCO [15], Garlic [4, 14]) employ postprocessing techniques to extend the capabilities of the sources they mediate over. In particular, they process a conjunction of query conditions or clauses, some of which are pushed to the source and the rest are executed while filtering the results of the source queries.

3.2 Query Decomposition

In [3] we showed how query-decomposition techniques in conjunction with postprocessing techniques can be used by mediators to process user queries with complex condition expressions over sources with limited query capabilities. The following example illustrates the need for the query-decomposition techniques.

Example 5. Consider the query (($author$ = "$Sigmund\ Freud$" $\vee$ $author$ = "$Carl\ Jung$") $\wedge$ $price < \$10$) to be posed to the Internet bookstore BarnesAndNoble.com. The query interface of BarnesAndNoble.com does not allow disjunctive condition specification. One might wonder if a postprocessing mediator on top of BarnesAndNoble.com could send the query ($price < \$10$) to the source and apply the author condition expression in filtering the source-query results. Unfortunately, BarnesAndNoble.com does not support queries that specify the price limit alone (queries that specify author and price are allowed). So, a mediator that simply tries to extend the source capabilities by way of postprocessing cannot support this query.

The mediator can decompose the problem query into two: one with the condition expression ($author$ = "$Sigmund\ Freud$" $\wedge$ $price < \$10$) and the other with the condition expression ($author$ = "$Carl\ Jung$" $\wedge$ $price < \$10$). Then, it can send these two queries to BarnesAndNoble.com. Both these queries are accepted by BarnesAndNoble.com. The results of the two source queries are combined (by taking the union) by the mediator and presented to the user. Thus, the mediator can support the query with the complex condition expression by employing the query-decomposition technique.

In [3] we present a scheme that can find feasible plans for queries with complex condition expressions over sources with limited capabilities. This scheme uses various transformation rules to identify different rewritings of the condition expression. It then finds parts of the rewritten condition expressions that can be evaluated at the source and generates a plan with the corresponding feasible source queries and the mediator operations (*union* and *intersection*) to combine the source-query results. For instance, the condition expression of the problem query of Example 5 is rewritten as ($author$ = "*Sigmund Freud*" $\wedge$ $price <$ \$10) $\vee$ ($author$ = "*Carl Jung*" $\wedge$ $price <$ \$10). Then, it identifies two feasible queries, each with a conjunctive condition expression, and creates a plan that takes the union of the two source-query results.

Some mediator systems like TSIMMIS [12] and Information Manifold [8] only accept conjunctive queries. That is, they do not accept the problem query at hand. DISCO [15] does accept disjunctive queries. However, DISCO does not explore the possibility of splitting the query condition into parts when looking for feasible plans. Only those options in which the source queries processes the entire condition expression, or no part of it, are considered. For instance, DISCO cannot support our problem query because BarnesAndNoble.com does not accept the query condition expression in its entirety and it also does not allow a query with an empty condition expression.

Garlic allows unrestricted query condition expressions. The condition expressions are transformed into *conjunctive normal form* (CNF) and then each clause in the CNF expression is considered for evaluation at the source [4]. If the source cannot evaluate a clause, it is evaluated by the Garlic mediator. If none of the clauses in the CNF expression can be evaluated at the source, Garlic attempts to download the entire source and process all the clauses at the mediator. In the case of the problem query of Example 5, the condition expression is already in CNF. Garlic notices that neither of the two clauses can be evaluated at the source, so Garlic attempts to download the entire source. Since BarnesAndNoble.com does not allow itself to be downloaded, Garlic fails to support the problem query.

4 Queries on Integrated Views

In the previous section we discussed how mediators can be employed to extend the sets of supported queries of sources with limited capabilities. There, each user query is targeted to a single source and the mediator for the source allows the user to pose queries that are more complex than what the source might accept. In this section we consider the use of mediators to provide integrated access to data from a *set* of sources and how the capability limitations of the data sources affect the processing of queries at such mediators.

In data-integration systems using the mediation architecture ([16]), mediators define integrated views in terms of source views and/or other integrated views at the mediator. User queries on integrated views are translated by mediators into source queries and mediator operations that combine the source-query results. The limited and heterogeneous capabilities of sources being integrated pose difficult challenges for mediators in generating efficient query plans involving answerable source queries. Along with the postprocessing and query-decomposition techniques discussed in Section 3, mediators integrating data from multiple sources use additional techniques like *passing bindings* ([11], [18]). The following example illustrates how mediators pass bindings across source queries to satisfy the binding requirements on certain attributes imposed by sources with limited capabilities.

Example 6. Consider a mediator with five data sources. Let the respective source views be $R_1(X, Y, Z)$, $R_2(X, Y, Z)$, $R_3(X, Y, Z)$, $R_4(Z, U)$, and $R_5(U, V, W)$. Let the mediator define the following two views: $M_1(X, Y, Z)$ is the union of R_1, R_2 and R_3; $M_2(X, Y, Z, U, V, W)$ is the join of$M_1(X, Y, Z)$, R_4 and R_5.

Users pose queries on the views of a mediator in a manner similar to submitting queries on source views. For instance, one can specify $M_2(x_1, y_1, Z, U, V, w_1)$ as a query to the mediator. The mediator translates this query into the following five source queries: $R_1(x_1, y_1, Z)$, $R_2(x_1, y_1, Z)$, $R_3(x_1, y_1, Z)$, $R_4(Z, U)$, $R_5(U, V, w_1)$. The results of the first three source queries are combined by a union operation, whose result is then combined with the results of the last two source queries by a join operation.

Suppose that the templates of the five sources are as follows: *buf*, *bfu*, *bff*, *fb*, *fub*. Based on these templates, we see that the first and the fourth source queries to R_1 and R_4 respectively are not supported by the sources. The mediator can handle the capability limitation of R_1 by sending the modified source query $R_1(x_1, Y, Z)$ and postprocessing the condition $Y = y_1$ on its result. The mediator deals with the binding requirement for U imposed by R_4 using the technique of passing bindings. That is, it can pass the bindings for the U attribute in $R_4(Z, U)$ from the results of the feasible source query $R_5(U, V, w_1)$.

In summary, the mediator can come up with the following plan for the query $M_2(x_1, y_1, Z, U, V, w_1)$: send $R_1(x_1, Y, Z)$ and filter its results by postprocessing the condition $Y = y_1$ into r_1; send $R_2(x_1, y_1, Z)$ and collect results into r_2; send $R_3(x_1, y_1, Z)$ and collect results into r_3; union r_1, r_2 and r_3 into s_1; send $R_5(U, V, w_1)$ and collect results into r_5; join s_1 and r_5 into s_2; pass bindings for the U attribute from s_2 to $R_4(Z, U)$, collect results into r_4; join s_2 with r_4 to get the overall result.

Notice that in the above example, when dealing with the capability limitation of R_4, the mediator first processes the query on R_5 and then passes bindings for the required attribute of R_4. In particular, when processing a query involving join views, the order in which the source queries are executed is important, as the results of some source queries can be used to pass bindings to other source queries. In the above example, the relative order of executing the source queries is not important as long as the query on R_5 is executed before the query on R_4 is executed. Of course, different query plans are obtained when considering different orders of the source queries. The various query plans may be different in their efficiency considerations. The mediator needs to identify not only feasible plans for user queries but also efficient plans.

In [18] we address the specific problem of generating efficient feasible plans for large join queries (i.e., user queries on join views at the mediator that are defined over large numbers of source views). A naive solution to this problem would be to consider all the orders of the source queries, identify those that yield feasible source queries, and select the one with the most efficient plan. Such a naive approach does not scale, and becomes impractical even for join views with moderately small fanout.

TSIMMIS mediators generate feasible plans for join queries by searching for feasible orders of source queries in a semi-naive fashion [11]. Although the algorithm does not blindly generate all the orders and check their feasibility, its running time is exponential in the size of the join queries (i.e., the number of join operands). Hence, the algorithm does not scale to find efficient feasible plans for large join queries. The Information Manifold [8] and Garlic [14] use similar search algorithms that also have exponential running times.

The exact solution to the problem of finding the optimal feasible plans is NP-Hard, as shown in [18]. So, it is unlikely that we can find an efficient algorithm that finds the optimal feasible plan for large join queries. We present algorithms in [18] that find very good feasible plans without requiring to find the optimal plans. These algorithms are very efficient, and in many cases they find optimal plans or near optimal plans. For one polynomial algorithm, we show bounded near optimality and for another algorithm we characterize a large class of scenarios in which the algorithm is guaranteed to find the optimal feasible plan.

5 Mediator Capabilities

Data sources with limited capabilities inform mediators about their supported-query sets. In Section 3, we have seen how mediators can extend the source capabilities by employing postprocessing and query-decomposition techniques. Later, in Section 4, we discussed how mediators provide integrated views that are defined on source views with limited query interfaces. The capability limitations of sources translate into the corresponding limitations of the mediator in terms of limiting the set of answerable queries on the integrated views. We define the capabilities of a mediator as the set of queries supported by the mediator on the integrated views it exports. The capabilities of a mediator can be computed from the capabilities of the sources it mediates.

Just as sources inform their users about their query capabilities, mediators should also describe to their users which queries they can support, so that users may know which queries to submit. In contemporary systems such as TSIMMIS [5, 11], Information Manifold [7, 8], Garlic [4, 14] and DISCO [6, 15], sources express their capabilities through a variety of mechanisms – query templates, capability records, and simple capability-description grammars. However, none of these systems computes the query-capabilities of mediators based on their source capabilities. Not having the mediator capabilities readily available makes it difficult to treat mediators as sources for other mediators. Furthermore, users have a difficult time understanding the set of supported mediator queries in these systems. Consequently, users must endure a frustrating trial-and-error approach, submitting queries that are rejected until finally hitting upon a query that is answered by the mediator.

Computing mediator capabilities can be simple in some cases. We may be able to combine all the restrictions of the sources in an easy manner to arrive at the capabilities of the mediator. For instance, consider an Internet mediator like the Compaq Shopping Guide for books [25], which provides a union view on top of a set of views provided by Web bookstores: Amazon.com and A1Books.com. Amazon.com provides a query form that allows users to search for books by specifying any one of author, title, format and subject attributes [21]. Another Web bookstore A1Books.com does not allow search by the format attribute. In addition, A1Books.com requires the specification of at least one of the author and title attributes [19]. The Compaq Shopping Guide is a mediator that provides a union view on top of these two (and other) Web bookstores. The combined restrictions of Amazon.com and A1Books.com suggests that the Compaq Shopping Guide can support queries that search for books by specifying the author or the title attribute. Queries specifying the subject attribute alone are not feasible because A1Books.com does not answer the corresponding queries. Queries satisfying the format attribute alone are again not feasible because A1Books.com does not support search by format.

In many cases, the computation of mediator capabilities is more complicated due to the rich variety of source-capability limitations (see Section 2) and the host of techniques mediators employ in processing queries. For instance, as we saw in Section 3, by using special techniques to postprocess the results of queries on limited-capability data sources, mediators can support much larger sets of queries. Thus, the capabilities we compute for mediators can be "stronger" than those of the underlying sources. For instance, both Amazon.com and A1Books.com do not support price specification in their queries, a mediator built on top of them can allow its users to specify price conditions that can be applied at the mediator when postprocessing the results of the Amazon.com and A1Books.com queries.

Even though in some cases it is not too complicated to compute mediator capabilities based on the capabilities of its sources, deriving mediator forms by hand is error-prone. Erroneously publishing a query form whose queries cannot be supported may lead to customer dissatisfaction when the apparent promises of the query form are not fulfilled. Similarly, erroneous computation that yields overly restricted forms leads to unnecessary loss of business as customers with the need to ask queries that are actually supported, but cannot express these queries in the overly restricted query forms, will visit other Web sources with their queries. Manual computation of mediator capabilities is also expensive, since each time a source changes its capabilities, we may have to update the mediator capabilities.

In [17] we have developed algorithms that compute the mediator capabilities automatically. Our algorithms work with the simple but powerful capability-description language presented in Section 2, and take into consideration various classes of mediators that employ a variety of techniques to extend the set of supported queries. The following example gives a flavor of the algorithms we present in [17].

Example 7. Let a mediator view $M(X, Y, Z)$ be defined as the union of two source views $R_1(X, Y, Z)$ and $R_2(X, Y, Z)$. Let R_1 have two templates: *bfo[*S_1*]* and *c[*S_2*]fb*. Let R_2 have two templates: *fbu* and *c[*S_3*]uf*.

To compute the templates for M, our algorithm considers four combinations of base-view templates: *bfo[*S_1*]* and *fbu*, *c[*S_2*]fb* and *fbu*, *bfo[*S_1*]* and *c[*S_3*]uf*, *c[*S_2*]fb* and *c[*S_3*]uf*. From the first combination, the algorithm computes the template *bbf* for M, because the mediator can allow optional specification of the Z attribute in user queries and postprocess the Z condition while sending source queries with no Z values. Similarly, postprocessing Z conditions on R_2 queries allows us to derive the template *c[*S_2*]bb* from the second combination. The other two combinations yield two more templates: *c[*S_3*]ff* and *c[*$S_2.S_3$*]fb* ($S_2.S_3$ is the intersection of S_2 and S_3). Thus, the mediator has four templates for the union view: *bbf*, *c[*S_2*]bb*, *c[*S_3*]ff* and *c[*$S_2.S_3$*]fb*.

5.1 Template Minimization

The number of templates computed for the mediator views can be very large when compared with the number of templates of the source views that are input to the computation. For instance, in the case of a mediator view that is a union of n source views with k templates each, we can end up with as many as k^n templates. More than the space it takes to store the mediator templates, a more serious concern here is that the query-capability description of the mediator may become so large as to make it difficult to ascertain whether a given query is answerable. A user or another mediator trying to figure out if a candidate query should be posed to a mediator would like a succinct specification of the mediator query capabilities. Fortunately, we may be able to reduce the size of the capability description significantly, based on the concept of minimizing template sets.

A set of templates is *minimal* if it does not contain redundant templates. A template in a set is *redundant* if every query allowed by that template is also allowed by at least one other template in the set. In [17] we provide a simple test that can be used to see if a template is redundant in a set. Here we illustrate the notion of template minimization by way of the following example.

Example 8. Consider the mediator view M of Example 7 for which we computed four templates: *bbf*, *c[*S_2*]bb*, *c[*S_3*]ff* and *c[*$S_2.S_3$*]fb*. We can deduce that *c[*S_2*]bb* is subsumed by *bbf*, because every query allowed by the former is also allowed by the latter. In other words, we can eliminate *c[*S_2*]bb* from the set of templates without shrinking the set of answerable queries. Similarly, *c[*S_2 . S_3*]fb* is subsumed by *c[*S_3*]ff*.

The removal of any one of the other two templates causes the set of answerable queries to shrink. For instance, the query $R(x_1, y_1, Z)$, where x_1 is not in S_3, is supported by the template *bbf* and is not supported by any of the other templates in the set. So, the removal of *bbf* shrinks the set of answerable queries. Similarly, we can see that *c[*S_3*]ff* is also not redundant. Thus, the given set of four templates can be minimized to two templates: *bbf* and *c[*S_3*]ff*.

Closely related to the idea of minimizing template sets by eliminating redundant templates is the idea of coalescing a set of simple templates into a smaller set of more powerful templates. For instance, consider a source view $R(X, Y, Z)$ with three templates: *bbf*, *ubf*, *fc[*S_1*]f*. Note that the first two templates can be coalesced into the more powerful single template *fbf*. That is, queries allowed by the first and the second template differ only in the specification of X. The first template requires the query to specify X while the second template disallows the specification of X. The *fbf* template allows queries that either specify X or optionally leave it unspecified. Since the adornments of the other attributes are the same, the first two templates can be replaced by the template *fbf*.

We note that coalescing not only helps in reducing the number of templates but also creates opportunities for eliminating other templates. For instance, in the original set of three templates it appears that no template is redundant because no template is subsumed by any other template. However, once we replace the first two templates by the single coalesced template *fbf*, we note that the template *fc[*S_1*]f* is subsumed by this coalesced template. Thus, we end up with a single template for $R(X, Y, Z)$ instead of three.

5.2 Capability Requirements

An interesting inverse problem to the one that computes the capabilities of a mediator based on the capabilities of its sources is as follows: given a query interface that is needed to be supported by the mediator on an integrated view, what are the capability requirements that must be satisfied by the source views on which the integrated view is defined?

A simple motivating example for the inverse problem comes from the context of a retail distributor trying to satisfy the information needs of its customers and in turn demanding its suppliers to satisfy corresponding information needs. When a customer to the retailer poses a query that needs to be answered, the retailer poses corresponding queries to its suppliers whose answers can then be combined to yield the answer to the customer query.

In some cases, the inverse problem may appear simple. For instance, in the case of the mediator discussed earlier that provides a union view over Web bookstores, one can argue that whatever form the customer of the mediator needs, the mediator can insist the sources to provide the same interface. At the outset, this may not be a good idea as it places a lot of hard requirements on the data sources and threatens their autonomy. Even if we are willing to compromise source autonomy it may not be prudent to push the requirements down the chain without making any effort to look for opportunities to soften the requirements. Suppose, for instance, in the mediator on Web bookstores, the end-user needs a query form that supports author, title and price conditions with at least one of author and title attributes to be filled out. The mediator may simply demand that Amazon.com and A1Books.com support the same query forms. A better solution would be to understand that the price condition can be postprocessed by the mediator and so Amazon.com and A1Books.com only need to support query interfaces that search by specifying at least one of author and title attributes. In fact, the current interfaces to the two Web bookstores support such search and so there is no need to do any additional work in providing new interfaces.

Minimizing the capability requirements that mediators put on their sources is also important when one considers the possibility that a source may provide data to more than one mediator. If every mediator demands its own set of requirements without making an effort to see if the currently supported source capabilities are sufficient or to identify the smallest extensions that are needed, the data source can be overwhelmed by the demands to support a large number of complex query interfaces.

In summary, we raise the following questions regarding the concept of deriving the capability requirements of sources from the capability requirements of mediators integrating them: given a mediator capability requirement, what is the minimum extension to the existing capabilities of the sources? when a source has multiple capability requirements for a view, can we coalesce the requirements in such a way that the source can support fewer query interfaces that satisfy all the requirements?

6 Content Information

Information about the content of data sources is often useful in processing queries over them. Content information can be useful both in extending the set of answerable queries and in answering queries efficiently, as illustrated by the following example.

Example 9. Consider a mediator view $M(X,Y,Z)$ defined as a join of two source views $R_1(X,Y)$ and $R_2(Y,Z)$. Let R_1 have the single template *bf* and let R_2 have the single template *c[S]f*, where S is $\{y_1, y_2, y_3\}$. Based on these two source templates, M has the single template *bc[S]f*. This template is derived for M because no matter which order the mediator executes the join, the Y value in the mediator query must be specified from the menu of choices S.

Query $M(x_1, y_1, Z)$ is answerable because it satisfies the template of M, while query $M(x_1, Y, Z)$ does not satisfy the template of M. However, the mediator may attempt to process $M(x_1, Y, Z)$ anyway. To process this query, the mediator identifies the need to perform the join of $R_1(x_1, Y)$ and $R_2(Y, Z)$. The mediator can perform this join by first executing $R_1(x_1, Y)$ and passing bindings for the

Y attribute in the queries to R_2. The set of Y values in the result of the query $R_1(x_1, Y)$ may turn out to be a subset of S. In this case, the query $M(x_1, Y, Z)$ can be answered successfully. However, we cannot determine whether the mediator query $M(x_1, Y, Z)$ is going to be answerable or not until run time, because its answerability depends on the contents of R_1.

Suppose the content description of R_1 indicates that the set of Y values at R_1 is $\{y_1, y_2\}$. Then, we can ascertain well before executing the query that $M(x_1, Y, Z)$ is an answerable query. In fact, we can even allow the more flexible query template *bff* for M.

Now, consider the query $M(x_1, y_4, Z)$. Once again, this query does not satisfy the template *bc[S]f* of M. According to the content description of R_1, Y has two values y_1 and y_2 at R_1. From this information, the mediator can determine without executing any source query that the result of $M(x_1, y_4, Z)$ is empty. In this case, the content description of R_1 not only helps the mediator avoid rejecting the query but also enables it to process this query efficiently by inferring the empty answer without incurring the cost of contacting the sources.

As noted in the above example, content descriptions indicating the set of values for an attribute at a source can be very useful in extending the set of feasible queries. Another kind of content description that is simple and quite useful is the expression of constraints, as illustrated by the following example.

Example 10. Suppose Amazon.com indicates that it has no books under the price of \$15. Consider a query that looks for books written by "Freud" costing less than \$10. Without having any content description of Amazon.com, the mediator will send the query that looks for all books written by "Freud" and postprocesses the price condition (recall that Amazon.com does not directly support queries specifying price conditions). Such expensive query execution can be avoided by the mediator if it has the content description that indicates that the answer to the query is empty.

To see how constraint information can help extend the set of answerable queries consider the query that looks for all books at Amazon.com that cost under \$10. Such a query cannot be processed directly at Amazon.com and post-processing techniques cannot help answer the query because Amazon.com does not answer the query with empty condition expression. Without the content description of Amazon.com, this query is unanswerable. Based on the content description, we can compute the answer to the query as the empty set and avoid rejecting the query.

We have indicated two simple kinds of content information about sources that can help in extending the set of answerable queries and also process queries on the sources more efficiently. A complete discussion of content description of sources and how they affect mediators built on top of them is not presented here. The interested reader is referred to [7], [9] and [10].

7 Source Unavailability

An important challenge in mediation systems is to deal with source unavailability. Particularly in contexts like the Internet, data sources become unavailable often and intermittently. When large numbers of sources participate in integrated views supported by mediators, queries on the integrated views often fail because some participating data source is unavailable at the time of executing the mediator query.

In order to deal with frequent failures of data sources, it is very useful to find alternative sources that provide the same information. During query execution, when a source failure is noticed, one of the alternative sources to the unavailable source can be substituted. Note that the alternative sources do not have to be identical replicas, they just need to be equivalent with respect to the source query at hand. Identifying alternative sources is directly related to our ability to obtain content description of data sources. In particular, we need inter-source content description.

If sources have identical query-processing capabilities, the challenge in dealing with source unavailability is limited to finding an efficient plan that takes advantage of alternative sources when it encounters unavailable sources. When alternative sources have unequal capabilities, substituting an alternative source for an unavailable source during query execution is complicated. In particular, the query to the source that is being substituted may not be accepted by the alternative source that is substituting the unavailable source.

A reasonable approach to dealing with alternative sources with unequal capabilities is to equate a source that is incapable of executing a query with a source that is unavailable. That is, when substituting an alternative source for an unavailable source during query execution, if the alternative source is incapable of executing the query slated for the unavailable source, we can simply consider the alternative source to be unavailable. We can then continue to look for an alternative source that is available and capable of answering the source query at hand. If we take this approach, we may end up with a situation in which none of a set of available alternatives to an unavailable source is capable of answering the source query and so we will not be able to execute the query plan. At the same time, it may be the case that the requirements of some of the alternatives to the unavailable source may indeed be satisfied by a different query plan (perhaps, using a different join order to generate bindings for the required attributes of the alternative sources). Thus, tying to execute the query using a single plan, by substituting alternatives for unavailable sources, may lead us to not execute the user query, even when it has a feasible plan that utilizes the available alternative sources.

We are currently researching issues surrounding the efficient query processing at mediators in the presence of source failures. Our research takes into consideration the possibility of alternative sources having unequal capabilities. We are also investigating problems arising from source unavailability and alternative sources with unequal capabilities when computing mediator capabilities.

In Section 5, and in [17], we have discussed the computation of mediator capabilities from those of its sources. What are the capabilities of a source with a set of alternatives that have unequal capabilities? One may choose to consider the capabilities of a source as the *lowest common denominator* of the capabilities of all its alternatives. Having ascertained the capabilities of the sources in this manner, we can then proceed to compute the mediator capabilities based on the source capabilities. Doing so is beneficial from the point of view of allowing the mediator to choose any alternative when a query satisfying a mediator template arrives, without being concerned about whether the choice of the alternative can make the mediator query infeasible. Thus, the mediator can choose the most efficient plan utilizing a flexible set of alternative sources. Another diametrically opposite approach is to take the union of the set of templates of all the alternative sources for a given source and use it as the set of source templates in the computation of mediator templates. In this case, when a query is posed that satisfies a mediator template, the mediator cannot freely choose any combination of alternative sources to answer the query because the specific combination the mediator chooses may not have the necessary source capabilities to be able to answer the mediator query. However, the advantage of this approach over the earlier one is that the mediator can have more flexible templates and so can allow the user to pose a larger set of feasible queries.

8 Conclusion

In query processing contexts like the Internet, data sources exhibit diverse and limited query capabilities. Many challenging problems arise when processing queries in such contexts. In this paper, we have briefly touched upon a few of the important problems.

We discussed how the capabilities of data sources can be described by a simple language based on query templates. We showed how the capabilities of data sources can be extended by mediators through postprocessing and query-decomposition techniques.

We considered data-integration systems that use mediators to process queries on integrated views that are defined on a set of source views. In these systems, we discussed how the limited capabilities of the data sources being mediated pose challenging problems to the mediator in processing queries on the integrated views. We need to treat mediators just like data sources so that we can build mediators on top of mediators. In this framework, the query capabilities of the mediators need to be computed from those of the sources they mediate. For the simple capability description language based on query templates, we discussed how mediator templates can be computed.

Finally, we showed how content description of sources can help not only in processing queries more efficiently but also in extending the set of queries supported. We also observed the interaction between source unavailability and capability limitations. In particular, we noted that in the presence of replicated sources with unequal capabilities, the problem of handling source unavailability becomes much harder.

References

1. M. Carey, et al. Towards heterogeneous multimedia information systems: the Garlic approach. *RIDE Workshop*, 1995.
2. S. Chawathe, et al. The TSIMMIS Project: Integration of Heterogeneous Information Sources. In *IPSJ Journal*, 1994.
3. H. Garcia-Molina, W. Labio, R. Yerneni. Capability-sensitive Query Processing over Internet Sources. In *Proc. ICDE Conference*, 1999.
4. L. Haas, D. Kossman, E. Wimmers, J. Yang. Optimizing Queries Across Diverse Data Sources. In *Proc. VLDB Conference*, 1997.
5. J. Hammer, H. Garcia-Molina, S. Nestorov, R. Yerneni, M. Breunig, V. Vassalos. Template-Based Wrappers in the TSIMMIS System. In *Proc. ACM SIGMOD Conference*, 1996.
6. O. Kapitskaia, A. Tomasic, P. Valduriez. Scaling Heterogeneous Databases and the Design of DISCO. *INRIA Technical Report*, 1997.
7. T. Kirk, A. Levy, Y. Sagiv, D. Srivastava. The Information Manifold. In *AAAI Symposium on Information Gathering in Distributed Heterogeneous Environment*, 1995.
8. A. Levy, A. Rajaraman, J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *Proc. VLDB Conference*, 1996.
9. A. Levy, M. Rousset. CARIN: A Representation Language Integrating Horn Rules and Description Logics. *AT&T Technical Report*, 1995.
10. A. Levy, Y. Sagiv. Constraints and Redundancy in Datalog. In *Proc. PODS Conference*, 1992.
11. C. Li, R. Yerneni, V. Vassalos, Y. Papakonstantinou, H. Garcia-Molina, J. Ullman. Capability Based Mediation in TSIMMIS. In *Proc. ACM SIGMOD Conference*, 1998.
12. Y. Papakonstantinou, H. Garcia-Molina, J. Ullman. Medmaker: A Mediation System Based on Declarative Specifications. In *Proc. ICDE Conference*, 1996.
13. Y. Papakonstantinou, H. Garcia-Molina, J. Widom. Object Exchange across Heterogeneous Information Sources. In *Proc. ICDE Conference*, 251–260, 1995.
14. Y. Papakonstantinou, A. Gupta, L. Haas. Capabilities-based Query Rewriting in Mediator Systems. In *Proc. PDIS Conference*, 1996.
15. A. Tomasic, L. Raschid, P. Valduriez. Dealing with Discrepancies in Wrapper Functionality. In *Proc. ICDCS Conference*, 1996.
16. G. Wiederhold. Mediators in the Architecture of Future Information Systems. In *IEEE Computer*, 25:38-49, 1992.
17. R. Yerneni, C. Li, H. Garcia-Molina, J. Ullman. Computing Capabilities of Mediators – Extended Version. `http://www-db.stanford.edu/pub/papers/medtemp.ps`, Stanford Technical Report, 1998.
18. R. Yerneni, C. Li, J. Ullman, H. Garcia-Molina. Optimizing Large Join Queries in Mediation Systems. In *Proc. ICDT Conference*, 1999.

19. A1Books.com Query Forms. http://www.a1books.com/cgi-bin/a1books/-search.cgi?act=advSearch/ December 15, 1998.
20. Amazon.com Web Site. http://www.amazon.com/ December 15, 1998.
21. Author-Title-Format-Subject Query Form for Amazon.com. http://www.amazon.com/exec/obidos/ats-query-page/ December 15, 1998.
22. ISBN Query Form for Amazon.com. http://www.amazon.com/exec/obidos/-subst/search/isbn.html/ December 15, 1998.
23. Keywords-Publisher-Date Query Form for Amazon.com. http://www.amazon.com/exec/obidos/subst/search/-publication-date.html/ December 15, 1998.
24. BarnesAndNoble.com Query Form. http://shop.barnesandnoble.com/-booksearch/search.asp? December 15, 1998.
25. Compaq Shopping Guide for Books. http://compaq.junglee.com/compaq/-searchbooks.shtml December 15, 1998.
26. Extensible Markup Language (XML) 1.0: Proposed Recommendation. http://www.w3.org/TR/REC-xml December 15, 1998.