Computing Capabilities of Mediators

Ramana Yerneni, Chen Li, Hector Garcia-Molina, Jeffrey Ullman
Department of Computer Science
Stanford University
{yerneni,chenli,hector,ullman}@cs.stanford.edu

Abstract

Existing data-integration systems based on the mediation architecture employ a variety of mechanisms to describe the query-processing capabilities of sources. However, these systems do not compute the capabilities of the mediators based on the capabilities of the sources they integrate. In this paper, we propose a framework to capture a rich variety of query-processing capabilities of data sources and mediators. We present algorithms to compute the set of supported queries of a mediator, based on the capability limitations of its sources. Our algorithms take into consideration a variety of query-processing techniques employed by mediators to enhance the set of supported queries.

1 Introduction

Many data integration systems (e.g., [1, 2, 3, 4, 6, 7]) use a *mediation* architecture [9] in which a mediator provides users with seamless access to information from heterogeneous sources. In mediation systems, one often encounters sources with diverse and limited query capabilities.

Contemporary mediator systems such as TSIMMIS [5, 6], Garlic [7], Information Manifold [4] and DISCO [3, 8] perform capability-based query processing. Sources express their capabilities in these systems 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 the supported source queries. 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.

In this paper, we present algorithms to precompute mediator capabilities automatically, so that users and other mediators know which queries are supported. In addition, we extend the types of source limitations that can be handled by existing mediation systems. For example, we handle attributes that can only be queried with values from a fixed menu of constants.

The World Wide Web is a prime example of a context where we need to handle many different types of source limitations. On the Web, data sources typically publish their query-processing capabilities through query forms. Users pose queries to data sources by filling out query forms and submitting them to the sources. For instance, the Web bookstore Amazon.com (`www.amazon.com`) provides a query form that allows users to search for books by specifying any one of author, title and subject attributes. Another Web bookstore A1Books.com (`www.a1books.com`) does not allow search by the subject attribute alone, while it provides search by the author or the title attribute.

Mediators built on Web sources frequently indicate their sets of supported queries through forms. For instance, the Junglee shopping guide for books (`compaq.junglee.com`) mediates across many Web bookstores including Amazon.com and A1Books.com. The Junglee mediator provides a query form that allows search by the title or the author attribute.

Computing mediator capabilities can be simple in some cases. For instance, the combined restrictions of Amazon.com and A1Books.com indicate that the Junglee 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 this query.

In many cases, the computation of mediator capabilities is more complicated due to the rich variety of source-capability limitations and the host of techniques mediators employ in processing queries. For instance, we show in Section 4 that by using special techniques to postprocess the results of queries on weaker 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.

Another complication arises when we consider mediator capabilities that depend on the contents of sources. In such cases, a mediator may define its capabilities with a "lower bound" that specifies which queries will always be answerable, and an "upper bound" indicating which queries will never be answerable. Queries "between" bounds may be answerable, depending on the contents of the sources at run time. We refer to mediators that handle upper and lower bounds as *dynamic mediators*. In Section 5, we discuss how dynamic mediators process queries and how their capabilities can be determined.

Deriving mediator forms by hand is error-prone. Manual computation of mediator capabilities is also expensive, since each time a source changes its capabilities, we may have to update the mediator capabilities. We need to develop algorithms to compute mediator capabilities automatically.

In this paper, we make the following specific contributions:

- *Framework*: We develop a framework for describing the restrictions on attribute specifications commonly found on the Web and in other heterogeneous data-integration contexts.
- *Algorithms*: We present algorithms for computing the query capabilities of mediators based on the capabilities of the sources they mediate.
- *Concise Description*: We provide strategies to condense the capability description of a mediator and enable efficient query-feasibility determination for users and other mediators that employ this mediator as a source.
- *Advanced Techniques*: We discuss advanced techniques that could be used by dynamic mediators to process queries beyond those that are feasible in conventional mediators.

2 Framework

In this section, we present our framework for describing the query capabilities of data sources and mediators. In our framework, each data source exports a set of relational views.[1] 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 2.1 Consider a source that exports a view $R(X, Y, Z)$. Let the set of tuples in source view R be $\{(x_1, y_1, z_1), (x_1, y_2, z_1), (x_2, y_2, z_2)\}$. Query $R(X, Y, z_1)$ results in $\{(x_1, y_1, z_1), (x_1, y_2, z_1)\}$, while query $R(X, y_1, Z)$ returns one tuple: (x_1, y_1, z_1). □

2.1 Mediator Views

A mediator integrates data from multiple sources and exports a set of integrated views. Each integrated view is defined in terms of source views and/or other integrated views. The set of operations used to define integrated views are *union*, *join*, *selection* and *projection*. We do not allow recursive views, so the mediator-view definitions form a directed acyclic graph (DAG). Since we allow the use of integrated views to define other integrated views, we can assume without loss of generality that each definition of a mediator view uses only one operator – union, join, selection or projection.

EXAMPLE 2.2 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 can pose queries on the views of a mediator in a manner similar to submitting queries on source views. For instance, one can specify $M_2(x_1, Y, Z, U, V, w_1)$ as a query to the mediator. □

2.2 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

[1] 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 [6] and XML (`www.w3.org/TR/REC-xml`). In fact, our interest in computing mediator capabilities is based on our work in the TSIMMIS project, which uses the OEM data model.

to be chosen from a menu of choices). We use attribute *adornments* to specify how the attributes of a view can participate in supported queries.

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.

EXAMPLE 2.3 Consider the source of Example 2.1 that exports the single view $R(X, Y, Z)$. 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, Y can be specified optionally while Z must be specified and must be chosen from $\{z_1, z_2\}$. These query capabilities are 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. □

We use the same mechanism of adorned templates to describe mediator capabilities. That is, the capabilities of a mediator are expressed as a set of templates on the views exported by the mediator. In this paper, we study the process of computing the templates of mediator views based on the templates of source views.

3 Templates for Simple Mediators

The set of answerable queries of a mediator is affected by the techniques used by the mediator in processing the queries posed to it. In this section, we start by considering the following simple query processing scheme: When a query is submitted, the mediator translates this query into a set of relevant source queries by transferring the query bindings; subsequently, the mediator combines the results of the source queries, based on the operations (union, join, selection, projection) appearing in the definition of the mediator view on which the query is submitted. In particular, simple mediators:

- do not perform any postprocessing other than that indicated in the view definitions;

- perform join operations locally, i.e., they do not pass bindings from one join operand to another.

The simple query-processing scheme we described above is likely to be supported by most mediators, so we choose it for our base case. We call mediators employing this query-processing scheme *simple mediators*. In Section 4, we discuss how mediators can employ additional postprocessing techniques and join methods to enhance their sets of answerable queries.

3.1 Union Views

We start by computing the set of templates for a union view. We present this computation in three steps. The first step deals with the simple case of a union view that has two base views, with each base view having exactly one template. Next, we show how to compute the set of templates with two base views, but with each base view having an arbitrary number of templates. Then, we describe the computation for a union view with an arbitrary number of base views, each having an arbitrary number of templates.

Union of Two Single-Template Base Views. For each attribute, we compute its adornment in the union-view template based on its adornments in the two base-view templates.[2] This computation is based on the mapping function presented in

[2] We assume here that both base views have the same set of attributes. In Section 6, we discuss the template computation for union views over base views that have different schemas.

	f	o[s_3]	b	c[s_4]	u
f	f	o[s_3]	b	c[s_4]	u
o[s_1]	o[s_1]	o[$s_1 \cap s_3$]	c[s_1]	c[$s_1 \cap s_4$]	u
b	b	c[s_3]	b	c[s_4]	-
c[s_2]	c[s_2]	c[$s_2 \cap s_3$]	c[s_2]	c[$s_2 \cap s_4$]	-
u	u	u	-	-	u

Table 1: Composition of Adornments

Table 1. Given the two adornments of an attribute in the two base-view templates, the table indicates the adornment of the attribute in the union-view template. For example, the entry in the table for the combination of f and b is b because the b adornment in one of the base-view templates forces the mediator to require that the attribute must be specified in the union-view query. Note that the mapping function of Table 1 is symmetric.

When the adornments of an attribute in the base-view templates involve menus of constants, we need to compute the resulting menu of constants. As indicated in Table 1, when only one of the base-view templates has a menu (with adornment o or c), this menu is copied over to the union-view template. When both the base-view templates have menus, we intersect the two menus.

In some cases, the base-view adornments of an attribute cannot be combined to arrive at a valid union-view adornment. Such cases are indicated by "-" in Table 1. For instance, consider the case of one base view having the b adornment and the other having the u adornment. No valid adornment can be computed in this case because the b adornment of the first base view forces the mediator to require that the attribute must be specified, while the u adornment of the other base view prevents the mediator from allowing the attribute to be specified in the union-view query.[3] Although not shown explicitly in Table 1, an attribute may also end up with an invalid adornment when computing its menu of constants by intersecting its menus from the base-view templates. In particular, the $c[s]$ entries in Table 1 are replaced by "-" when s is empty. Note that the $o[s]$ entry should be replaced by u when s is empty.

During the computation of the union-view template, if any attribute is determined to have the invalid adornment, we declare that no union-view template can be computed from the two base-view templates.

Union of Two Base Views with Multiple Templates. For each pair in the cross product of the template sets of the two base views, we compute a template for the union view based on the process described above. As noted earlier, in some cases no union-view template can result from a pair of base-view templates. Accordingly, the number of templates for a union view over two base views with template sets T_1 and T_2 varies from zero to $|T_1| \times |T_2|$.

Union of Multiple Base Views with Multiple Templates. We compute the templates of the union view by considering two base views at a time. That is, if the union view has n base views, we invoke the method of computing templates for a union view with two base views $(n-1)$ times. Note that the associativity and symmetry of the mapping function presented in Table 1 allow us to carry out the computation in this simple manner.

EXAMPLE 3.1 Let a mediator view M be defined as the union of three source views $R_1(X, Y, Z)$, $R_2(X, Y, Z)$ and $R_3(X, Y, Z)$. Let R_1 have two templates: *bff* and *ffb*, let R_2 have the single template *fbf*, and let R_3 have two templates: *ffc[s_1]* and *c[s_2]ff*.

When computing the templates of $R_1 \cup R_2$, we consider two combinations of base-view templates: *bff* for R_1 and *fbf* for R_2; *ffb* for R_1 and *fbf* for R_2. Based on the first combination, we compute the template *bbf*, and the second combination yields the template *fbb*.

Now, for $(R_1 \cup R_2) \cup R_3$, four combinations of templates are considered and they result in the following four templates for M: *bbc[s_1]*, *fbc[s_1]*, *c[s_2]bf* and *c[s_2]bb*. Notice that it may be possible to "collapse" this set of templates into a smaller set that still captures the same capability information. For instance, we can eliminate two of the four templates for M and keep only two templates: *fbc[s_1]* and *c[s_2]bf*. In Section 3.4 we discuss how template sets can be reduced to arrive at concise capability description. □

3.2 Join Views

As noted earlier, a simple mediator processes a query on a join view by first transferring the query bindings to the base views, and then joining the results of the base-view queries. Since the join-view query processing is similar to the union-view query processing, the computation of templates for join views is similar to that of union-view templates. However, unlike in the case of a union view, the attributes of a join view may not appear in all of its base views. So, the computation of attribute adornments in a join-view template is slightly different from that in a union-view template.

[3] Recall that we are dealing with simple mediators in this section. In the next section, we will see how mediators use postprocessing techniques to arrive at a valid adornment of b when the base-view adornments are b and u.

Join of Two Single-Template Base Views. For all nonjoin attributes, we copy over their adornments from the base-view templates (each nonjoin attribute appears in exactly one of the base views). For each join attribute, the adornment computation employs the same mapping function (see Table 1) that is used when computing union-view templates.

Join of Two Base Views with Multiple Templates. As in the union case, for each pair in the cross product of the sets of base-view templates, we compute a join-view template.

Join of Multiple Base Views with Multiple Templates. Once again, as in the union case, we consider two base views at a time. If the join view has n base views, we invoke the method of computing templates for a join view with two base views $(n-1)$ times.

3.3 Selection and Projection Views

When processing a query on a selection view, the mediator copies it over into a query on the underlying base view and applies the selection predicate on the results of the base-view query. Therefore, the set of base-view templates are simply copied over as the set of selection-view templates.

A query on a projection view is translated into a query on the underlying base view by simply leaving the hidden attributes (those that are in the base view but not in the projection view) unspecified. If any of the hidden attributes has a b or c adornment in a base-view template, the translated query does not match this base-view template. Therefore, we only create a template for the projection view whenever the base-view template has the f, o or u adornments for the hidden attributes. The created projection-view template simply copies over the adornments for each of the projected attributes from the base-view template.

3.4 Concise Capability Description

The number of templates computed for mediator views can be very large. 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. A large number of templates makes 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 eliminating *redundant* templates. Informally, a template in a set is redundant if every query allowed by it is also allowed by at least one other template in the set. A complete discussion of the problem of identifying redundant templates and eliminating them is beyond the scope of this paper. Here, we briefly describe a simple technique that helps us eliminate redundant templates.

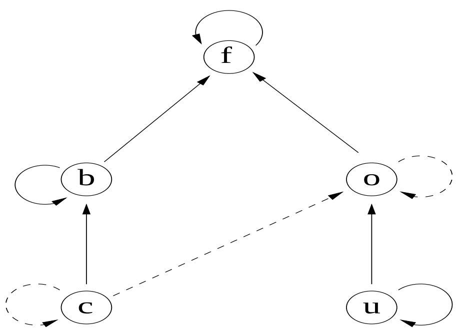

Figure 1: The Adornment Graph

First, we develop the notion of comparing the restrictiveness of two attribute adornments. The relative restrictiveness of the various kinds of attribute adornments is captured by the graph of Figure 1. The set of nodes in the graph are the five adornments: f, o, b, c and u. A solid arc in the graph from node n_1 to node n_2 represents the fact that the adornment of n_1 is at least as restrictive as that of n_2. Broken arcs originate and terminate with adornments that have constant sets associated with them (i.e., c and o adornments). A broken arc from node n_1 to node n_2 represents the fact that the adornment of n_1 is at least as restrictive as the adornment of n_2 if the constant set associated with the former is a subset of the constant set associated with the latter. For instance, $c[s_1]$ is at least as restrictive as $o[s_2]$ if s_1 is a subset of s_2.

Note that the adornment-restrictiveness relationship represented by Figure 1 is transitive. For instance, c is at least as restrictive as f because it is at least as restrictive as b, which in turn is at least as restrictive as f. Based on the relative restrictiveness of adornments discussed above, we specify the following simple test for identifying redundant templates in a set.

Subsumption Test: A template T is subsumed by another template T' if for every attribute X the adornment of X in T is at least as restrictive as the adornment of X in T' (based on Figure 1).

As illustrated in the following example, we use the Subsumption Test to identify redundant templates in a set of templates (those subsumed by other templates in the set).

EXAMPLE 3.2 Suppose a view has the following set of templates: $\{bff, fbf, ffb, c[s_1]fb, ubo[s_2]\}$. Based on the Subsumption Test defined above, we deduce that *c*$[s_1]$*fb* is subsumed by *bff*, and *ubo*$[s_2]$ is subsumed by *fbf*. Thus, the given set of five templates can be reduced to $\{bff, fbf, ffb\}$. □

3.5 Summary Remarks

Our algorithm for computing mediator templates has time complexity that is exponential in the input size (the number of source-view templates and the size of the mediator-view definitions). However, the exponential time complexity is not an important concern because all this computation is performed "offline," when the mediator is formed on a set of sources (not when a query is being processed).

We have presented the machinery to compute the templates of a single mediator view starting from the templates of its base views. We can extend this machinery in a straightforward manner to compute the templates of all the views in a mediator view DAG by considering them in topological order.

4 Advanced Query-Processing Techniques in Mediators

In this section, we consider some techniques employed by mediators to support more queries than those supported by the simple mediators of the previous section. We start by presenting examples that illustrate two important techniques used by mediators, *postprocessing* and *passing bindings*, and how they impact template computation.

EXAMPLE 4.1 Let M be a mediator view defined as the union of two source views $R_1(X,Y,Z)$ and $R_2(X,Y,Z)$. Suppose R_1 has the template *bfu*, and R_2 has the template *buf*. In the case of a simple mediator, we compute the single template *buu* for M. Based on this template, $M(x_1,Y,Z)$ is a feasible query, while $M(x_1,y_1,z_1)$ is not.

If the mediator can postprocess the results of queries on the underlying views, then it can support more queries. For instance, it can support the query $M(x_1,y_1,z_1)$ by first invoking the feasible queries $R_1(x_1,y_1,Z)$ and $R_2(x_1,Y,z_1)$, and then filtering the results of these queries with respect to the conditions on the Y and the Z attributes. In particular, it can apply the condition $(Z = z_1)$ on the result of $R_1(x_1,y_1,Z)$ and the condition $(Y = y_1)$ on the result of $R_2(x_1,Y,z_1)$. The union of the postprocessed results of source queries gives the answer to the query $M(x_1,y_1,z_1)$. Thus, the ability of the mediator to postprocess the results of underlying queries can enhance the set of feasible queries supported by the mediator. □

	f	**o**$[s_3]$	**b**	**c**$[s_4]$	**u**
f	f	f	b	c$[s_4]$	f
o$[s_1]$	f	f	b	c$[s_4]$	f
b	b	b	b	c$[s_4]$	b
c$[s_2]$	c$[s_2]$	c$[s_2]$	c$[s_2]$	c$[s_2 \cap s_4]$	c$[s_2]$
u	f	f	b	c$[s_4]$	f

Table 2: Union with Postprocessing

EXAMPLE 4.2 Let a mediator view M be the join of two source views $R_1(X,Y,Z)$ and $R_2(Z,U,V)$. Suppose R_1 has the single template *bfb*, and R_2 has the single template *fub*. In the case of a simple mediator, M has the single template *bfbub*. For instance, $M(x_1,Y,z_1,U,v_1)$ is a feasible query while $M(x_1,y_1,Z,U,v_1)$ is not.

If the mediator can perform join operations by passing bindings from one join operand to the next (i.e., it can perform *bind joins* [11]), the query $M(x_1,y_1,Z,U,v_1)$ can be answered. The mediator can first execute the query $R_2(Z,U,v_1)$ and pass bindings for the Z attribute. For each value of Z, say z_i, in the result of $R_2(Z,U,v_1)$, the mediator can invoke the query $R_1(x_1,y_1,z_i)$. The answer to the query on M is obtained from the results of all these R_1 queries and the R_2 query. Thus, the ability to perform join operations by passing bindings enhances the set of queries a mediator can support. □

4.1 Union Views

As before, the computation of union-view templates can be described in three steps. In fact, the only difference between this computation and the one in Section 3.1 is in the first step, where we compute the template of a union view defined over two base views, each with a single template.

Union of Two Single-Template Base Views. Using the same notation as before, we define a new mapping function for the computation of an attribute's adornment in the union-view template from the attribute's adornments in the two base-view templates. The new mapping function is shown in Table 2.

The essential difference between the mapping function of Table 2 and the mapping function used in Section 3.1 is in the treatment of the u and the o adornments. When a base-view adornment is u, the mediator can invoke a query on this view without specifying a value for this attribute and then optionally support a value specified by the union-view query for this attribute in the postprocessing step. Therefore, the u adornment is treated the same way as the f adornment. In a similar way, when a base-view template has the o adornment for an attribute, and a value specified

	f	o[s_3]	b	c[s_4]	u
f	f	f	f	c[s_4]	f
o[s_1]	f	f	f	c[s_4]	f
b	b	b	b	c[s_4]	b
c[s_2]	c[s_2]	c[s_2]	c[s_2]	c[$s_2 \cap s_4$]	c[s_2]
u	f	f	f	c[s_4]	f

Table 3: Join by Passing Bindings

by the union-view query for this attribute is not one of the menu constants associated with the *o* adornment, the mediator can execute a query on the base view without specifying any value for this attribute and check for the value given by the union-view query in the postprocessing step. Thus, *o* is also treated as *f*.

4.2 Join Views

When processing a query on a join view over a set of base views, since the attribute values returned from one base-view query can be used to satisfy the binding requirements of the subsequent base-view queries, the order in which the base-view queries are considered is important. Accordingly, the template computation for join views is presented in four steps. The first step considers a join sequence of two base views, each with one template. Then we deal with a join sequence of two base views, each with an arbitrary number of templates. Next we handle a join sequence of an arbitrary number of base views, each with an arbitrary number of templates. Finally, in the fourth step, we compute the templates of a join view by considering all the sequences of its base views.

Join Sequence of Two Single-Template Base Views. As before, for all nonjoin attributes we copy over their adornments from the base-view templates. The mapping function for computing the adornment of a join attribute is presented in Table 3. The adornment of the first base-view is listed on the left and the adornment of the second base-view is listed on the top of the table. Because the mediator can perform joins by passing bindings, the case of a *b* adornment for the second base view is similar to the case of an *f* adornment. The mediator passes the required binding for the second base view from the result of the query on the first base view.

Join Sequence of Two Base Views with Multiple Templates. As in the union case, we repeatedly invoke the method that handles single-template base views and compute the set of templates of the join sequence. That is, for each combination of base-view templates, we obtain a template for the join sequence.

Join Sequence of Multiple Base Views with Multiple Templates. We associate left-to-right the base views in the join sequence (note that Table 3 is associative, but not symmetric). That is, if the join sequence has n base views, we call the method of computing templates for a join sequence of two base views $(n-1)$ times.

Join View of Multiple Base Views with Multiple Templates. We consider all the possible sequences of the base views, and for each sequence, we invoke the above method to compute a set of templates. For a join view with n base views, we call this method $n!$ times. We take the union of the $n!$ resulting sets of templates to arrive at the set of templates for the join view.

EXAMPLE 4.3 Let a mediator view M be defined as the join of three base views $R_1(X,Y,Z)$, $R_2(Z,U,V)$ and $R_3(V,W)$. Let R_1 have two templates: *fbf* and *bfb*, let R_2 have two templates: *bfb* and *fbf*, and let R_3 have the single template *fb*.

We consider a total of six sequences. For the sequence $\langle R_1, R_2, R_3 \rangle$, we end up with four templates: *fbffbb*, *fbfbfb*, *bfbfbb* and *bfbbfb*. For the sequence $\langle R_3, R_2, R_1 \rangle$, the templates are: *fbbffb*, *fbfbfb*, *bfbffb* and *bffbfb*. Continuing in this manner, we can compute another four sets of four templates each, based on the remaining four sequences. The union of these six sets of templates yields a set of 24 templates for M. After minimizing this set of templates, we end up with the following three templates for M: *fbfffb*, *bfbffb*, *bffbfb*.

Without the ability to perform joins by passing bindings, the set of templates for the mediator view M would be limited to: *fbbfbb*, *fbfbfb*, *bfbfbb* and *bfbbfb*. Notice that these four templates are more restrictive than the three we obtained for a mediator that passes bindings. In particular, the set of queries covered by the three templates of M provided by the mediator through bind joins is a strict superset of the set of queries covered by the four templates obtained through local joins. For instance, the query $M(X, y_1, Z, U, v_1, w_1)$ is feasible only if the mediator passes bindings. □

4.3 Selection and Projection Views

For selection views, the template computation is quite different from that of Section 3.3. We do generate a selection-view template corresponding to each base-view template. However, we do not simply copy over the adornments from the base-view templates to the selection-view templates.

EXAMPLE 4.4 Let M be a selection view with $R(X,Y,Z)$ as its base view and $(X < x_1)$ as the selection condition. Let R have a template *bfu*. According to the template computation in Section 3.3, M will have the *bfu* template.

The *bfu* template of M does not allow a query like $M(x_2, Y, z_1)$. However, the mediator can make use of its postprocessing abilities to support such a query. Given $M(x_2, Y, z_1)$, the mediator can first process the feasible base-view query $R(x_2, y_1, Z)$ and then filter the results of this query with the condition $(Z = z_1)$. Therefore, M can have the more flexible template *bff*.

Now, suppose that the selection predicate on M is $(X = x_1)$ instead of $(X < x_1)$. The *bff* template of M precludes a query like $M(X, y_1, Z)$. However, the mediator can infer from the selection-view predicate that it can translate $M(X, y_1, Z)$ into $R(x_1, y_1, Z)$, a feasible query on the base-view. So, it can support the query $M(X, y_1, Z)$. To reflect this ability to support "additional" queries on M, the template of M is changed to *fff*. Thus, a mediator that performs postprocessing can have selection-view templates that are much more flexible than their corresponding base-view templates. □

Base View Adornment	Sel. Attribute Adornment	Nonsel. Attribute Adornment
f	f	f
o$[s_1]$	f	f
b	f or b	b
c$[s_1]$	f or c$[s_1]$	c$[s_1]$
u	f	f

Table 4: Selection with Postprocessing

The new rules for the computing selection-view templates are based on the mapping function given in Table 4. We consider two cases for each attribute in the selection view: (i) the selection predicate specifies a value for the attribute; (ii) the selection predicate does not specify a value for the attribute. In both cases, by employing postprocessing operations at the mediator, base-view adornments of o and u are converted to the selection-view adornment of *f*. In addition, the b adornment is also converted to the less restrictive *f* adornment if a value for the attribute can be inferred from the selection predicate. A c adornment is converted into an *f* adornment if the selection predicate specifies a value for the attribute that is in the constant set associated with the attribute's adornment in the base-view template. If the inferred value is not in the set of constants of the base-view template, we simply copy over the c adornment to the selection-view template.

The computation of templates for projection views is similar to that of Section 3.3, except that u and o adornments in the base-view templates are treated as *f* adornments. That is, we copy over the *f*, b and c adornments of the projected attributes from the base-view templates to the corresponding projection-view templates, while the u and o for the projected attributes are changed to the *f* adornment in the projection-view templates. As before, we do not derive a projection-view template from a base-view template that has the b or the c adornment for a hidden attribute.

5 Dynamic Mediators

So far, we have seen how the templates of a mediator can be computed in order to specify the set of queries answerable by the mediator. Given that computation, a query is supported by the mediator if it satisfies one of the mediator templates. However, as illustrated by Example 5.1, it may not be necessary for a query to satisfy a template in order for it to be answerable.

EXAMPLE 5.1 Consider a mediator view M 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 the computation described in Section 4, M has the single template *bc[s]f*. Query $M(x_1, y_1, Z)$ is answerable because it satisfies the template of M.

Consider the query $M(x_1, Y, Z)$. This query does not satisfy the template of M. However, the mediator may attempt to process the query anyway. It can perform a bind join by processing the query $R_1(x_1, Y)$ 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. □

Based on Example 5.1, we note that the answerability of a user query is not entirely determined by checking it against the templates computed according to the methods of Sections 3 and 4. If the query satisfies some template, it is guaranteed to be answerable, otherwise, its answerability depends on the current state of the data in the source views. For instance, the current state of R_1 in Example 5.1 may help make the query answerable. Then again, the state of R_1 may be such that the query cannot be answered. Mediators attempting to execute queries that are not guaranteed to be answerable, to determine query answerability in a data-dependent manner at run time, are called *dynamic* mediators.

5.1 Conservative and Liberal Templates

Dynamic mediators execute queries that are not guaranteed to be feasible, with the hope of answering them in a data-dependent manner. However, as illustrated by Example 5.2, it is sometimes possible to determine that a query is infeasible without attempting to execute it.

EXAMPLE 5.2 Consider M, defined as a join of $R_1(X,Y)$ and $R_2(Y,Z,U)$. Let R_1 have the single template *bf* and let R_2 have the single template *c[s]fb*, where s is $\{y_1, y_2, y_3\}$. Then, M has the template *bc[s]fb*. This template of M indicates that the query $M(x_1, y_1, Z, u_1)$ is definitely answerable. The query $M(x_1, Y, Z, u_1)$ does not satisfy the template of M. However, it is answerable if the set of Y values at R_1 is a subset of s. The query $M(x_1, Y, z_1, U)$ also does not satisfy the template *bc[s]fb*. We can determine that this query is not going to be answerable, irrespective of the set of Y values at R, because it does not specify a value for U. Blindly trying to execute it in a dynamic mediator results in an expensive way of finding out that the query is infeasible. □

Example 5.2 showed that there may be situations in which we can determine that a given query is not going to be answerable irrespective of the state of the data in the sources. It is desirable to be able to specify a set of templates such that if none of them is satisfied by a query, we can ascertain that the query is infeasible, without trying to execute it futilely. Such templates specify an "upper bound" to the set of queries that can be answered by a dynamic mediator, while the templates computed in the previous section form the "lower bound." We call the first kind *liberal* templates and the second kind *conservative* templates.

Each view has a set of conservative templates and a set of liberal templates. Given a query on a view, we ascertain that the query is answerable if it satisfies at least one of the conservative templates of the view, and the query is not answerable if it does not satisfy any of the liberal templates of the view. If a query does not satisfy any conservative templates but satisfies at least one liberal template, then a dynamic mediator executes the query in a data-dependent manner. For instance, in Example 5.2, we can specify a conservative template *bc[s]fb* and a liberal template *bffb* for the mediator view M. Based on these templates, we can determine that query $M(x_1, y_1, Z, u_1)$ is guaranteed to be answerable; query $M(x_1, Y, Z, U)$ is guaranteed to be unanswerable; query $M(x_1, Y, Z, u_1)$ may be answerable depending on the contents of R_1.

5.2 Computing Liberal Templates

Typically, in the case of a source view, the liberal templates of the view are the same as the conservative templates. When computing the templates of derived views at a mediator, the liberal templates tend to diverge from their conservative counterparts. The algorithms of Section 4 yield conservative templates for mediator views. That is, we start with conservative templates of base views and obtain conservative templates of derived views.

With small changes to the algorithms of Section 4, we can compute the liberal templates. For the selection, projection and union views, we use the same algorithms to compute the liberal templates of derived views starting with the liberal templates of their base views. However, the computation is slightly different in the case of join views.

	f	**o**$[s_3]$	**b**	**c**$[s_4]$	**u**
f	f	f	f	f	f
o$[s_1]$	f	f	f	f	f
b	b	b	b	b	b
c$[s_2]$	c$[s_2]$	c$[s_2]$	c$[s_2]$	c$[s_2]$	c$[s_2]$
u	f	f	f	f	f

Table 5: Liberal Templates for Join Views

For a join view, we start with the liberal templates of the base views and compute the corresponding liberal templates of the join view, in a manner that is quite similar to that of Section 4.2. The only difference is the use of a new mapping function that combines the attribute adornments of the base-view templates. The new mapping function is given in Table 5. This mapping function is similar to the one used in Section 4.2, except in the case of the c adornment in the second base-view template. In this case, when computing the liberal template of the join view, the mediator allows a more flexible adornment because it can try to get the appropriate constant required by the second base view from the result of the query on the first base view. That is, it can optimistically treat the c adornment in the second base-view template as if it is the f adornment. For instance, when the first base-view adornment is f and the second base-view adornment is c, the liberal adornment for the join-view is f (instead of c in the conservative computation of Section 4.2). Also, note that when both the base view adornments are c, the resulting c adornment in the liberal template for the join-view has the same constant set as the c adornment in the first base view (instead of the intersection of the constant sets in the two base-view templates).

6 Output Restrictions

The computation of mediator templates discussed so far assumes that all the attributes of a view are returned in response to any query on that view. 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 by specifying the subject attribute of the desired set of books, and Amazon.com does not return the subject attribute when answering queries.

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 introduced in Section 2: *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.

The computation of mediator templates in the presence of output restrictions can be undertaken by modified versions of the algorithms of Sections 3 and 4. Only the mapping functions used by the algorithms have to be extended to deal with the new adornments. In particular, no postprocessing operations can be performed on attributes that are not returned in source query results. Note that even necessary operations like joining on such attributes are prevented. All these considerations can be reflected in a new set of mapping functions for the algorithms presented so far. Due to space limitations, we do not present the new mapping functions here (please refer to the extended version of the paper [10] for them). The introduction of the new attribute adornments forces us to also reconsider the adornment graph of Figure 1, which is the basis for identifying and eliminating redundant templates. The new graph is given in the extended version of the paper.

Recall that in Sections 3 and 4 we assumed that the base views of a union view have the same schema. With the help of the new attribute adornments, we can handle the case of union views with heterogeneous base-view schemas. When encountering heterogeneous base-view schemas, we can simply treat the situation as if all the base-views have the same schema by adding the missing attributes to each base-view schema. For the newly added attributes, we specify the *u'* adornments in the templates of the base views. To illustrate, consider a mediator view $M(X, Y, Z)$ defined as a union of $R_1(X, Y)$ and $R_2(X, Z)$. Let R_1 have the template *bf'* and let R_2 have the template *bf'*. We introduce the missing attribute Z into the template of R_1 and the missing attribute Y into the template of R_2. Both Z and Y have the *u'* adornment in the templates of R_1 and R_2, respectively. From the resulting *bf'u'* and *bu'f'* templates, we can compute the appropriate *bu'u'* template for M (based on the new mapping function for computing union-view templates, presented in [10]).

7 A Case Study

To verify that our capability-description framework makes sense in practice, we explored the Web, where many limited-capability sources are found. In particular, we wished to determine if the adornments we developed in our framework were adequate in describing the query capabilities of sources and mediators. We also wanted to know how many templates were typically required to describe sources and mediators. Capability-based query processing could become unwieldy if large numbers of templates were required. Therefore, it is important to check if in the case of representative sources and mediators there would be an explosion of templates.

7.1 Data Sources

We considered two Web bookstores in our case study: Amazon.com (`www.amazon.com`) and BarnesAndNoble.com (`shop.barnesandnoble.com`).

Amazon.com supplies the following query forms:

- Form 1: At least one of author, title, subject and format attributes must be specified. The format attribute has a menu of choices.
- Form 2: The ISBN attribute must be specified.
- Form 3: At least one of keywords, publisher and publication date attributes must be specified.

The results of queries to Amazon.com include the following attributes: author, title, ISBN, publisher, date, format, price and shipping info. In particular, the subject and keywords attributes do not appear in the answers.

The query capabilities of Amazon.com are described by the templates in Table 6. The capabilities offered by each query form are captured by one or more templates. In Table 6, the first four templates capture Form 1, the fifth template captures Form 2, while the last three templates correspond to Form 3. Note that for simplicity of presentation

author	title	format	subject	KW	ISBN	pub	date	price	ship
b	f	o	f'	u'	u	u	u	u	u
f	b	o	f'	u'	u	u	u	u	u
f	f	c	f'	u'	u	u	u	u	u
f	f	o	b'	u'	u	u	u	u	u
u	u	u	u'	u'	b	u	u	u	u
u	u	u	u'	f'	u	b	f	u	u
u	u	u	u'	b'	u	f	f	u	u
u	u	u	u'	f'	u	f	b	u	u

Table 6: Templates of Amazon.com

we did not show the menu of choices attached to the *o* and *c* adornments of the format attribute. The menu specified by Amazon.com has "Hard cover", "Paperback", etc.

BarnesAndNoble.com has two query forms:

- Form 1: At least one of author, title and keywords attributes must be specified. In addition, the format, subject, price and age range attributes can be specified optionally. These four attributes have menus of choices.
- Form 2: The ISBN attribute must be specified.

The output attribute set of BarnesAndNoble.com is the same as that of Amazon.com. That is, the source does not return the subject, keywords and age range attributes in its answers. The capabilities of BarnesAndNoble.com are described by the templates in Table 7. The first three templates describe the capabilities of the first form, while the last template captures the second form.

7.2 A Bookstore Mediator

We considered a bookstore mediator that provides a union view over the above two source views. As suggested in Section 6, we handled the heterogeneous union of the two source views by adding the age attribute to the templates of Amazon.com with the *u'* adornment. Moreover, we assumed that our bookstore mediator employs postprocessing techniques to extend the set of feasible union-view templates, as discussed in Section 4. Accordingly, we computed a total of 22 templates for the mediator view. Our algorithm actually considered 32 (8×4) pairs of source-view templates and successfully generated union-view templates in the case of 26 pairs. However, there were 4 duplicates among the 26 resulting templates. Then, we employed the techniques discussed in Section 3.4 to identify and eliminate 14 redundant templates and ended up with a concise set of 8 mediator templates (see Table 8).

From the set of 8 templates for the mediator, 4 query forms can be derived. Corresponding to the first template in Table 8, the bookstore mediator has a query form that requires the specification of the ISBN attribute. Corresponding to the second template, we created a query form that requires the keywords to be specified. Next, we created a single query form that corresponds to the third and fourth templates. This query form requires that at least one of author and title attributes must be specified along with an optional specification of the subject field from a menu of choices. Finally, the last four templates were combined into a single query form that requires that at least one of author and title attributes and at least one of publisher and date attributes must be specified.

Note that, the theoretical maximum number of forms for our bookstore mediator is 32 (because each combination of the base-view templates could have resulted in a mediator template, and each mediator template could end up as a separate query form). The fact that in our case study we obtained 4 query forms for the mediator suggests that using the techniques presented in this paper, we may be able to compute manageable sets of query forms for mediators on Web sources.

7.3 Observations

Our case study demostrates that the capability-description framework we introduced in Section 2 is well suited to describe the capabilities of Web sources and mediators. We note that sometimes more than one template is needed to describe a query form. However, the number of templates required to describe a form is typically small. We also observe that, in general, it is more difficult to derive forms from templates than vice versa.

In the course of our experiments, we noticed that many Web sources change their query forms frequently. In fact, the data for our case study as presented above is valid as of March 1, 1999. It is unlikely that the Web sources we considered in our case study will retain the same forms a few months later. In our experience, sources change their Web forms many times in a short period of time (a few times a year). Building mediators on such evolving sources poses special challenges.

author	title	format	subject	KW	ISBN	pub	date	price	ship	age
f	b	o	o’	f’	u	u	u	o	u	o’
b	f	o	o’	f’	u	u	u	o	u	o’
f	f	o	o’	b’	u	u	u	o	u	o’
u	u	u	u’	u’	b	u	u	u	u	u’

Table 7: Templates of BarnesAndNoble.com

author	title	format	subject	KW	ISBN	pub	date	price	ship	age
f	f	f	u’	u’	b	f	f	f	f	u’
f	f	f	u’	b’	f	f	f	f	f	u’
b	f	f	o’	u’	f	f	f	f	f	u’
f	b	f	o’	u’	f	f	f	f	f	u’
b	f	f	u’	f’	f	b	f	f	f	u’
f	b	f	u’	f’	f	b	f	f	f	u’
f	b	f	u’	f’	f	f	b	f	f	u’
b	f	f	u’	f’	f	f	b	f	f	u’

Table 8: Concise Set of Templates for the Bookstore Mediator

In particular, manual generation of query forms for mediators on such evolving sources becomes very difficult because whenever sources change their query forms, mediator capabilities have to be reassessed. Automatic computation of mediator capabilities based on the techniques presented in this paper can be very helpful to mediation systems involving frequently changing sources.

8 Conclusion

In data-integration systems, it is important to describe the capabilities of mediators so that they can be used as easily (by end users as well as other applications) as base sources are. Many contemporary integration systems have not computed and exported mediator capabilities, thus making it hard for them to be useful in scalable applications involving networks of mediators and sources. In some situations, mediator capabilities are manually computed and specified. Manual computation is error prone and becomes unwieldy when dealing with large numbers of evolving sources whose query capabilities change frequently.

In this paper, we provided the machinery for automatically computing the capabilities of mediators based on the capabilities of the sources they integrate. We proposed a capability-description framework with a rich set of attribute adornments to describe a variety of query-processing limitations of sources and mediators. We discussed various classes of mediators based on the query processing techniques they employ, and presented algorithms for the computation of their capabilities. We conducted experiments using Web sources and studied issues surrounding the adequacy of our capability-description framework and the effectiveness of our algorithms for computing mediator capabilities.

References

[1] Y. Arens, C. Knoblock, W. Shen. Query Reformulation for Dynamic Information Integration. *Journal of Intelligent Information Systems*, 6(2/3):99-130, 1996.

[2] M. Genesereth, A. Keller, O. Duschka. Infomaster: An Information Integration System. *Proc. SIGMOD Conference*, 1997.

[3] O. Kapitskaia, A. Tomasic, P. Valduriez. Scaling Heterogeneous Databases and the Design of Disco. *INRIA Technical Report*, 1997.

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

[5] C. Li, R. Yerneni, et al. Capability-Based Mediation in TSIMMIS. *Proc. SIGMOD Conference*, 1998.

[6] Y. Papakonstantinou, H. Garcia-Molina, J. Ullman. Medmaker: A Mediation System Based on Declarative Specifications. *Proc. ICDE*, 1996.

[7] Y. Papakonstantinou, et al. Capabilities-Based Query Rewriting in Mediator Systems. *Proc. PDIS Conference*, 1996.

[8] A. Tomasic, L. Raschid, P. Valduriez. Dealing with Discrepancies in Wrapper Functionality. *Proc. ICDCS*, 1996.

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

[10] R. Yerneni, C. Li, et al. Extended Version: Computing Capabilities of Mediators. `www-db.stanford.edu/~ yerneni/pubs/ccmev.ps`.

[11] R. Yerneni, C. Li, et al. Optimizing Large Join Queries in Mediation Systems. *Proc. ICDT*, 1999.