Computing Capabilities of Mediators

Ramana Yerneni, Chen Li, Hector Garcia-Molina, Jeffrey Ullman
Department of Computer Science
Stanford University
contact: yerneni@cs.stanford.edu

Abstract

In data-integration systems, the queries supported by a mediator are affected by the query-processing limitations of the sources being integrated. Existing mediation systems 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 and present algorithms to compute the set of mediator-supported queries 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, and they yield concise capability descriptions. By computing mediator query capabilities and representing them in the same way as those of data sources, we enable mediators to be used by other mediators, and we also make it easier for end users to know in advance which mediator queries are feasible.

1 Introduction

Many data integration systems [2,4,5,7,10,12,18,25] use a *mediation* architecture [27] 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 (e.g., an author must be provided when searching for books; house prices can be queried but not retrieved). Typically, sources inform the mediator which queries they can support, so that the mediator can construct execution plans that use feasible source queries. In turn, a mediator should inform *its* users which queries it can support, so that the users may know which queries to submit. In this paper, we study the problem of computing the set of supported queries of a mediator, based on the capabilities of its sources.

Contemporary mediator systems such as TSIMMIS [5,8,16], Information Manifold [11,12,13], Garlic [4,23,24], and DISCO [9,25] 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 provides a query form that allows users to search for books by specifying any one of author, title and subject attributes [32]. Another Web bookstore A1Books.com does not allow search by the subject attribute alone, while it provides search by the author or the title attribute [30]. Wrappers that provide access to such sources would have the same type of limited capabilities.

Mediators built on Web sources frequently indicate their sets of supported queries through forms. For instance, the Yahoo shopping guide for books [37] mediates across many Web bookstores including Amazon.com and A1Books.com. The Yahoo 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, consider a mediator like the Yahoo shopping guide for books, which provides a union view on top of a set of views provided by Web bookstores. We may be able to combine all the restrictions of the sources in an easy manner to arrive at the capabilities of the mediator. The combined set of restrictions of Amazon.com and A1Books.com suggests that the Yahoo 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 5 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 6, 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. We also model restrictions on attribute output in the results of queries.
- *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. A concise capability description of a mediator enables 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.

We start with a mediator specification that defines integrated views in terms of source views. We assume that differences in ontologies, vocabularies, and formats used by sources have been resolved. In particular, if two sources share an attribute name, we assume that the attributes are equivalent (i.e., wrappers take care of any differences). Related research [17,22] suggests ways to deal with ontology and format differences. The problem we are addressing in this paper specifically deals with the computation of mediator capabilities, after the mediator views and their definition in terms of source views are identified. Note that we use a flat attribute-specification model for query capabilities (like relational-view templates) for simplicity of exposition. We believe that the main ideas of the paper are easily adaptable to nested attribute models like OEM [8] and XML [36].

2 Related Work

In this section, we briefly discuss the approaches taken by contemporary mediation systems to deal with query-processing capability limitations of the data sources they integrate. TSIMMIS [5,8] uses *query templates* [8] to describe source capabilities. In a template, each attribute can be adorned as *bound*, *free* or bound to a single fixed *constant*. The mediator defines integrated views on the source views using the MSL language [21]. An MSL view definition specifies the content of the view, but does not indicate which queries can be supported using the view. When a user submits a query, the mediator translates the query into a logical plan by expanding the views specified in the query, and, if possible, generates a *feasible* physical plan based on the source templates [16]. At the time of submitting a query, the user does not know if the query is actually executable, i.e., if a feasible plan will be found. The TSIMMIS mediator does not describe its capabilities to the user, so the user has to submit queries in a trial-and-error scenario.

The Information Manifold [11, 12, 13] uses *capability records* to describe source capabilities. Each source has a capability record that indicates the input attributes of a 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 $op \in \{\leq, <, \neq, =\}$. Capability descriptions in the Information Manifold are limited to data sources. That is, as in TSIMMIS, an Information Manifold mediator does not compute and export its capabilities to users and other mediators using it as a source.

Garlic [4,23,24] 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 will be answerable by the source. As with TSIMMIS and Information Manifold, Garlic mediators do not have capability descriptions. That is, users posing queries to a Garlic mediator have no idea about the set of answerable queries.

Other integration systems, like DISCO [9,25], SIMS [2] and Araneus [18], use different languages to describe source capabilities. Again, mediators in these systems do not compute and export their capabilities.

The NAIL! system [20] studied the problem of generating query plans for integrated views in the presence of source-capability restrictions. However, NAIL! only considered a limited set of restrictions expressed as attribute adornments, *free* and *bound*. Also, NAIL! did not precompute the adornments for integrated views. In contrast, we consider a richer set of attribute adornments and precompute feasible adornments of integrated views. The LDL system [6,26] used the notion of query forms. Given a set of query forms, it precomputed skeleton query plans for them. When a query was submitted, if it fit one of the precomputed query forms, the LDL system could instantiate the corresponding skeleton plan to execute the query. The results of our paper complement their work by identifying the set of mediator query forms for which skeleton plans may be precomputed.

Related research has explored heterogeneous database schema integration and data translation (e.g., [1], [3], [19]) and capability-based query optimization (e.g., [23], [29]). Our work here, on computing mediator capabilities, complements those efforts.

3 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 3.1 Consider a source that exports a view $R(X, Y, Z)$. Let the set of tuples in source view R be $\{(x_1, y_1, z_1), (x_1, y_2, z_1), (x_2, y_2, z_2)\}$. 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 include $R(x_1, Y, z_1)$, $R(x_1, Y, Z)$ and $R(X, Y, Z)$. $\square$

3.1 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 of the mediator. The set of operations used to define integrated views are *union*, *join*, *selection* and *projection*.

[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 ([8]). In fact, our interest in computing mediator capabilities is based on our work in the TSIMMIS project, which uses the OEM data model.

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 3.2 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. $\square$

3.2 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 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.

EXAMPLE 3.3 Consider the source of Example 3.1 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 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. □

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.

As noted earlier, templates indicate restrictions on attribute specification in queries to sources and mediators. In Section 7, we show how templates can also be used to describe restrictions on the set of attributes returned when mediators and sources answer queries. We discuss there the effect of output restrictions on computing the capabilities of mediators. Until then, we do not consider output restrictions in the computation of mediator templates, as presented in Sections 4 and 5.

4 Templates for Simple Mediators

In this section, we describe the process of computing the capabilities of a mediator based on the capabilities of its sources. That is, we compute templates for a mediator based on the templates of its data sources.

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, we make the following assumptions:

- Mediators do not apply any postprocessing conditions, other than those listed explicitly in the view definitions.
- Mediators 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 5, we discuss how mediators can employ additional postprocessing techniques and join methods to enhance their sets of answerable queries.

4.1 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 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.

[2]We assume here that both base views have the same set of attributes. In Section 7, we discuss the template computation for union views over base views that have different schemas.

	f	**o**$[S_{21}]$	**b**	**c**$[S_{22}]$	**u**
f	f	o$[S_{21}]$	b	c$[S_{22}]$	u
o$[S_{11}]$	o$[S_{11}]$	o$[S_{11} \cap S_{21}]$	c$[S_{11}]$	c$[S_{11} \cap S_{22}]$	u
b	b	c$[S_{21}]$	b	c$[S_{22}]$	-
c$[S_{12}]$	c$[S_{12}]$	c$[S_{12} \cap S_{21}]$	c$[S_{12}]$	c$[S_{12} \cap S_{22}]$	-
u	u	u	-	-	u

Table 1: Mapping Function for Combining Adornments from Two Templates.

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

[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.

to carry out the computation in this simple manner.

EXAMPLE 4.1 Let a mediator view $M(X,Y,Z)$ 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 4.5 we formally discuss how template sets can be minimized to arrive at concise capability description. □

4.2 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, there is one major difference: The attributes in a join view may not appear in all its base views. This distinction between join attributes and nonjoin attributes leads to a slightly different way of computing the attribute adornments in join-view templates.

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) used by the union-view template computation.

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.

EXAMPLE 4.2 Let a mediator view $M(X, Y, Z, U, V, W)$ 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 *bff*, let R_2 have two templates: *bfb* and *fc[S_1]f*, and let R_3 have the single template *fb*. The join of R_1 and R_2 results in four templates: *fbbfb*, *fbfc[S_1]f*, *bfbfb* and *bffc[S_1]f*. When we join in R_3, the four templates are modified as: *fbbfbb*, *fbfc[S_1]fb*, *bfbfbb* and *bffc[S_1]fb*. □

4.3 Selection Views

When processing a query over 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.

EXAMPLE 4.3 Let $M(X, Y, Z)$ be a selection view with $R(X, Y, Z)$ as its base view and $(X < x_1)$ as the selection predicate. Let R have the template *fbu*. Then, M also has the template *fbu*. For instance, one can pose a query on M by specifying a value for the attribute Y, like $M(X, y_1, Z)$. The mediator can copy over this query on M into the query $R(X, y_1, Z)$, and apply the predicate $(X < x_1)$ on the result of the query on R. □

Note that we can be "smarter" than simply repeating the adornments of the base-view templates in the selection-view templates. For instance, if a source does not allow queries specifying a particular attribute (i.e., it has the *u* adornment for this attribute), the mediator could allow more flexible queries by processing extra predicates on that attribute, along with processing the selection-view predicate on the results of the corresponding source queries. Such special postprocessing of source-query results represents more mediator processing than is considered by the base case of our simple mediators. We return to this topic in Section 5 after we have completed discussing the fundamentals of template computation.

4.4 Projection Views

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.

EXAMPLE 4.4 Consider a projection view $M(X, Z)$ defined over the base view $R(X, Y, Z)$. Let R have the following templates: *fbu*, *bfu*. The first template of R does not yield any template of M because the hidden attribute Y has the b adornment. From the second template of R, we compute the *bu* template of M. That is, M can support queries that bind the X attribute and that leave the Z attribute unspecified. □

4.5 Concise Capability Description

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, our 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. Informally, a template in a set is *redundant* if it is strictly more restrictive than another template in the set.

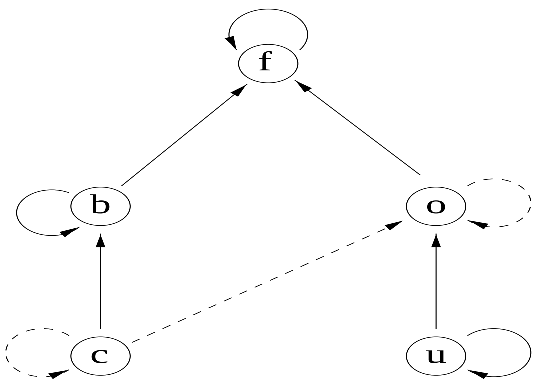

Figure 1: The Adornment Graph.

The degree of restrictiveness of the various 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 formal 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).

EXAMPLE 4.5 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 minimized to $\{bff, fbf, ffb\}$. □

Note that the Subsumption Test can also be used to check if a query q is feasible with respect to a template T. From q, we construct a template T_q wherein each attribute specified in the query has a c adornment with a menu of one constant (the value specified in the query), and each attribute left unspecified in the query has the u adornment. Then, we declare that q is feasible with respect to T if and only if T_q is subsumed by T. In a similar way, a query to a mediator is considered feasible if the mediator has a template with respect to which the query is feasible. Note that the process of minimizing template sets does not compromise the correct determination of the set of feasible queries of mediators.

4.6 Efficient Template Computation

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). Minimizing template sets of intermediate views, during the process of computing mediator templates, helps make the process quite efficient in practice.

4.7 Templates for Graph of Views

So far, 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 at a mediator. In particular, we obtain the templates of the views exported by the mediator and thus ascertain the set of queries supported by the mediator.

In the view graph of the mediator there is an edge from V_1 to V_2 whenever V_2 is used in defining V_1. Recall that the view graph at a mediator is a directed acyclic graph with the source views having no outgoing edges. Based on the techniques described above, we compute the templates of all the views in the graph respecting the order constraints imposed by the edges of the graph. That is, we choose a topological sorting of the views and compute their adornments in that order. During the process of computing the templates of views in the mediator view graph, we eliminate redundant templates. Minimization of template sets during view-graph template computation leads to an efficient process of computing the templates of all the mediator views.

Whenever the mediator specification is modified, we can recompute the templates of only those views that are affected by the changes. A mediator view is affected by the changes if its definition is altered or if its definition contains a view that is affected by the changes.

5 Advanced Query-Processing Techniques in Mediators

In this section, we consider some techniques used 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 5.1 Let a mediator view $M(X, Y, Z)$ be the union of two source views $R_1(X, Y, Z)$ and $R_2(X, Y, Z)$. Suppose R_1 has the single template *bfu*, and R_2 has the single 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. □

EXAMPLE 5.2 Let a mediator view $M(X, Y, Z, U, V)$ 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* instead of the simple *local joins* of Section 4), 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 from the result of this query to the query on R_1. That is, 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. □

Based on the techniques discussed above, we will re-examine the computation of mediator-view templates to identify the larger sets of feasible queries. Specifically, we consider union, join, selection and projection views, and describe how their template computation differs from that of the previous section.

5.1 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 4.1 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 4.1 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

	f	**o**$[S_{21}]$	**b**	**c**$[S_{22}]$	**u**
f	f	f	b	c$[S_{22}]$	f
o$[S_{11}]$	f	f	b	c$[S_{11} \cap S_{22}]$	f
b	b	b	b	c$[S_{22}]$	b
c$[S_{12}]$	c$[S_{12}]$	c$[S_{12}]$	c$[S_{12}]$	c$[S_{12} \cap S_{22}]$	c$[S_{12}]$
u	f	f	b	c$[S_{22}]$	f

Table 2: Mapping Function for Union with Postprocessing.

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 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*.

EXAMPLE 5.3 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 *o[*S_2*]fb*. Let R_2 have the single template *fbu*.

To compute the templates for M, we consider two combinations of base-view templates: *bfo[*S_1*]* and *fbu*, *o[*S_2*]fb* and *fbu*. From the first combination, we arrive at the template *bbf* for M. The second combination results in the template *fbb*.

Note that if the mediator does not perform postprocessing, we will only have one template for M: *bbu*. Besides, this template is more restrictive than the *bbf* template that we obtained above by using mediator postprocessing. □

5.2 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

	f	**o**$[S_{21}]$	**b**	**c**$[S_{22}]$	**u**
f	f	f	f	c$[S_{22}]$	f
o$[S_{11}]$	f	f	f	c$[S_{11} \cap S_{22}]$	f
b	b	b	b	c$[S_{22}]$	b
c$[S_{12}]$	c$[S_{12}]$	c$[S_{12}]$	c$[S_{12}]$	c$[S_{12} \cap S_{22}]$	c$[S_{12}]$
u	f	f	f	c$[S_{22}]$	f

Table 3: Mapping Function for Join by Passing Bindings.

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 can pass 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 5.4 Let a mediator view $M(X, Y, Z, U, V, W)$ 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. $\square$

5.3 Selection and Projection Views

For selection views, the template computation is quite different from that of Section 4.3. We do generate a selection-view template corresponding to each of its base-view templates. However, we do not simply copy over the adornments of attributes from the base-view templates to the selection-view templates.

EXAMPLE 5.5 Let $M(X, Y, Z)$ 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 4.3, the mediator has the template *bfu* for M.

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, let us 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. Thus, 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*. In this way, a mediator that performs postprocessing can have selection-view templates that are much more flexible than their corresponding base-view templates. $\square$

Base-View Adornment	Selection-Attribute Adornment	Nonselection-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: Mapping Function for Selection with Postprocessing.

Base-View Adornment	Projection-View Adornment
f	f
o$[S_1]$	f
b	b
c$[S_1]$	c$[S_1]$
u	f

Table 5: Mapping Function for Projection with Postprocessing.

The new rules for the computation of the 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 4.4, except that *u* and *o* adornments in the base-view templates are treated as *f* adornments. As shown in Table 5, 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.

6 Dynamic Mediators

In this section, we explore the possibility of mediators being able to support user queries that do not satisfy the templates of the mediator views (as computed in Section 5). We also discuss extensions to our capability-description language to deal with restrictions on the output of attributes in query results.

6.1 Data Dependencies

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 6.1, it may not be necessary for a query to satisfy a template in order for it to be answerable.

EXAMPLE 6.1 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 the computation described in Section 5, 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 the query $M(x_1, Y, Z)$ can be answered successfully. □

Based on Example 6.1, 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 4 and 5. 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 6.1 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.

6.2 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 6.2, it is sometimes possible to determine that a query is infeasible without attempting to execute it.

EXAMPLE 6.2 Consider a mediator view $M(X, Y, Z, U)$ defined as a join of two source views $R_1(X, Y)$ 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 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. $\square$

Example 6.2 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 6.2, 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 state of the data in R_1.

6.3 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

	f	**o**$[S_{21}]$	**b**	**c**$[S_{22}]$	**u**
f	f	f	f	f	f
o$[S_{11}]$	f	f	f	f	f
b	b	b	b	b	b
c$[S_{12}]$	c$[S_{12}]$	c$[S_{12}]$	c$[S_{12}]$	c$[S_{12}]$	c$[S_{12}]$
u	f	f	f	f	f

Table 6: Mapping Function for Liberal Join-View Templates.

templates tend to diverge from their conservative counterparts. The algorithms of Section 5 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 5, 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.

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 5.2. 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 6. This mapping function is similar to the one used in Section 5.2, 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 5.2). 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).

6.4 Content Information and Query Answerability

We have seen how dynamic mediators may be able to answer queries even though the queries do not satisfy the conservative templates. However, in such situations the mediators cannot ascertain the answerability of the queries until run time. Here we briefly explore mediators that use information about the contents of the source views in order to ascertain query answerability statically. There are many interesting issues related to content description of sources and how it affects mediators built on top of them ([12],[14], [15]). Due to space limitations, we limit our discussion to the following few remarks.

We note that content descriptions have an important subtle difference from capability descriptions. For instance, in Example 6.1, declaring that R_2 has a template *c[S]f*, where S is $\{y_1, y_2, y_3\}$, indicates that queries to R_2 must specify one of these three constants for the Y attribute. The *c[S]f* template of R_2 does not mean that R_2 has no tuple with a Y value that is not in S. In fact, R_2 could very well have a tuple with a Y value y_4, and would refuse to answer the query $R_2(y_4, Z)$. On the other hand, if we were to have a content description of R_2 that declares that the set of Y values at R_2 is S, a mediator can answer the query $R_2(y_4, Z)$ by returning an empty result.

Content descriptions of sources help determine query feasibility at compile time in dynamic mediators. For instance, in Example 6.1, if we have a content description of R_1 indicating that the set of Y values at R_1 is a subset of S (the constant set for the Y attribute in the template of R_2), M can have the more flexible template *bff*, instead of the earlier template *bc[S]f*. Note that there is a clear difference between such content-based templates and the liberal templates discussed earlier. The new *bff* template is not really an "upper" bound. If a query satisfies this new template, it is guaranteed to be answerable. In this sense, the *bff* template is a more flexible conservative template, not a liberal template.

7 Output Restrictions on Attributes

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 [32] 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 introduced in Section 3: *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.

The computation of mediator templates in the presence of output restrictions can be undertaken by modified versions of the algorithms in Sections 4, 5 and 6. Basically, only the mapping functions used by the algorithms have to be extended to deal with the new adornments.

7.1 Template Computation for Simple Mediators

Here we present the new mapping functions for the algorithm of Section 4, which computes templates for simple mediators.

7.1.1 Union Views

The new mapping function for computing union-view templates in simple mediators is given in Table 7. Note that the new mapping function considers all ten attribute adornments (as opposed to the five considered by Table 1. However, since the simple mediators do not perform any postprocessing the entries in the new mapping function are quite similar to those of Table 1.

The mediator can output a union-view attribute if both base views output the attribute. The mediator cannot output a union-view attribute if neither base view outputs the attribute. As shown in Table 7, we adopt the policy that if an attribute is output by only one of the base views, the mediator outputs the attribute in the union view. When taking the union of the results of the two base-view queries, the mediator can introduce $NULL$ values for the missing attribute values. Such a policy is adopted in practice by Web mediators like the Yahoo shopping guide for books [37].

Recall that in Section 4, we assumed that all the base views of a union view have the same

	f	**o**[S_{21}]	**b**	**c**[S_{22}]	**u**	**f'**	**o'**[S_{23}]	**b'**	**c'**[S_{24}]	**u'**
f	f	o[S_{21}]	b	c[S_{22}]	u	f	o[S_{23}]	b	c[S_{24}]	u
o[S_{11}]	o[S_{11}]	o[$S_{11} \cap S_{21}$]	c[S_{11}]	c[$S_{11} \cap S_{22}$]	u	o[S_{11}]	o[$S_{11} \cap S_{23}$]	c[S_{11}]	c[$S_{11} \cap S_{24}$]	u
b	b	c[S_{21}]	b	c[S_{22}]	-	b	c[S_{23}]	b	c[S_{24}]	-
c[S_{12}]	c[S_{12}]	c[$S_{12} \cap S_{21}$]	c[S_{12}]	c[$S_{12} \cap S_{22}$]	-	c[S_{12}]	c[$S_{12} \cap S_{23}$]	c[S_{12}]	c[$S_{12} \cap S_{24}$]	-
u	u	u	-	-	u	u	u	-	-	u
f'	f	o[S_{21}]	b	c[S_{22}]	u	f'	o'[S_{23}]	b'	c'[S_{24}]	u'
o'[S_{13}]	oS_{13}]	o[$S_{13} \cap S_{21}$]	c[S_{13}]	c[$S_{13} \cap S_{22}$]	u	o'[S_{13}]	o'[$S_{13} \cap S_{23}$]	c'[S_{13}]	c'[$S_{13} \cap S_{24}$]	u'
b'	b	c[S_{21}]	b	c[S_{22}]	-	b'	c'[S_{23}]	b'	c'[S_{24}]	-
c'[S_{14}]	c[S_{14}]	c[$S_{14} \cap S_{21}$]	c[S_{14}]	c[$S_{14} \cap S_{22}$]	-	c'[S_{14}]	c'[$S_{14} \cap S_{23}$]	c'[S_{14}]	c'[$S_{14} \cap S_{24}$]	-
u'	u	u	-	-	u	u'	u'	-	-	u'

Table 7: Mapping Function for Simple Union and Join with Output Restrictions.

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 simply treat the situation as if all the base-views have the same schema by "adding" the adornment of *u'* for the missing attributes to each base-view template. 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 single template *bf* and let R_2 have the single template *bu*. We introduce the adornment of *u'* for the missing attribute Z in the template of R_1 and the adornment of *u'* for the missing attribute Y into the template of R_2.

7.1.2 Join Views

As in Section 4, the computation of join-view templates uses the same mapping function as the computation of union-view templates. Therefore, the new mapping function for computing join-view templates in simple mediators in the presence of output restrictions is the also given by Table 7.

7.1.3 Selection Views

As in Section 4.3, each base-view template yields a corresponding selection-view template with the adornments copied over. This direct mapping is an obvious extension of the one in Section ssec-selectionview, as it copies over the new adornments as well as the old adornments.

	f	**o**$[S_{21}]$	**b**	**c**$[S_{22}]$	**u**	**f'**	**o'**$[S_{23}]$	**b'**	**c'**$[S_{24}]$	**u'**
f	f	f	b	c$[S_{22}]$	f	f	o$[S_{23}]$	b	c$[S_{24}]$	u
o$[S_{11}]$	f	f	b	c$[S_{22}]$	f	f	o$[S_{23}]$	b	c$[S_{24}]$	u
b	b	b	b	c$[S_{22}]$	b	b	c$[S_{23}]$	b	c$[S_{24}]$	-
c$[S_{12}]$	c$[S_{12}]$	c$[S_{12}]$	c$[S_{12}]$	c$[S_{12} \cap S_{22}]$	c$[S_{12}]$	c$[S_{12}]$	c$[S_{12} \cap S_{23}]$	c$[S_{12}]$	c$[S_{12} \cap S_{24}]$	-
u	f	f	b	c$[S_{22}]$	f	f	o$[S_{23}]$	b	c$[S_{24}]$	u
f'	f	f	b	c$[S_{22}]$	f	f'	o'$[S_{23}]$	b'	c'$[S_{24}]$	u'
o'$[S_{13}]$	o$[S_{13}]$	o$[S_{13}]$	c$[S_{13}]$	c$[S_{13} \cap S_{22}]$	o$[S_{13}]$	o'$[S_{13}]$	o'$[S_{13} \cap S_{23}]$	c'$[S_{13}]$	c'$[S_{13} \cap S_{24}]$	u'
b'	b	b	b	c$[S_{22}]$	b	b'	c'$[S_{23}]$	b'	c'$[S_{24}]$	-
c'$[S_{14}]$	c$[S_{14}]$	c$[S_{14}]$	c$[S_{14}]$	c$[S_{14} \cap S_{22}]$	c$[S_{14}]$	c'$[S_{14}]$	c'$[S_{14} \cap S_{23}]$	c'$[S_{14}]$	c'$[S_{14} \cap S_{24}]$	-
u'	u	u	-	-	u	u'	u'	-	-	u'

Table 8: Mapping Function for Union Views with Postprocessing and Output Restrictions.

7.1.4 Projection Views

As in Section 4.3, we attempt to create a projection-view template from each base-view template. Whenever a base-view template has a *b*, *c*, *b'* or *c* adornment for a hidden attribute, we do not derive a projection-view template. Otherwise, we copy over the adornment for each projected attribute from the base-view templates to the corresponding projection-view templates. That is, the new mapping function for computing projection-view templates in simple mediators is a direct extension of the one in Section 4.4.

7.2 Template Computation for Complex Mediators

Here we present the new mapping functions for the algorithm of Section 5, which computes templates for mediators that perform postprocessing and bind joins.

7.2.1 Union Views

The new mapping function for computing union-view templates in postprocessing mediators is given in Table 8. Notice that in the first five rows and the first five columns (the top-left quadrant) of Table 8, both base views output the attribute in their results. Therefore, the mediator can perform postprocessing as in Section 5 and so the entries are identical to those of Table 2. The entries in the last five rows and the last five columns (the bottom-right quadrant) of Table 8 are similar to Table 1 because both base views are suppressing the attribute, and so the mediator cannot perform any postprocessing on this attribute. In the rest of Table 8, the mediator can postprocess on the attribute returned from one of the base views. Consequently, the entries are less flexible than the top-left quadrant and are more flexible than the bottom-right quadrant. For instance, the combination of *u* and *o* yields *f* (top-left quadrant); the combination of *u'* and *o'* yields *u'* (bottom-right quadrant); and the combination of *u* and *o'* yields *o* (top-right quadrant).

As in Section 7.1.1, we can handle union views over base views with heterogeneous schemas by introducing missing attributes with *u'* adornments. Regarding the output of attributes in a union view, we adopt the policy that the mediator can output an attribute if at least one of the two base views outputs the attribute.

We observe that a "smart" mediator can alleviate the output restrictions on attributes that are already specified in the queries. For instance, a *b'* adornment for an attribute in a template indicates that a value must be specified for the attribute in a feasible query, and that the attribute will not be output in the result of the query. In this case, the mediator can simply copy the input value from the query to the query result, thus allowing the attribute to be present in the output. Hence, the *b'* adornment can be treated in the same way as the less restrictive *b* adornment. Similarly, the *c'* adornment may be treated as the *c'* adornment. Notice that the cases of *f'* and *o'* where an input value is specified in query can also lead to the attribute being present in the output. However, the mediator cannot guarantee that the attribute will be present when the adornment is *f'* or *o'*. That is, it cannot convert *f'* to *f* and *o'* to *o*. Note that the entries in Table 8 do not actually reflect the more flexible adornments in the cases when both the base views do not output an attribute.

7.2.2 Join Views

Table 9 presents the new mapping function for computing the adornment of a join attribute in a join sequence of two base-views with single templates. The *f'* and the *o'* adornments of the base-view templates are treated in a similar fashion to the *b'* and the *c'* adornments. The reason is that when a base-view query does not output the attribute, the attribute cannot be left unspecified in the base-view query because local joins are not possible due to the absence of the join attribute in the result of the base-view query. Note that if the join attribute has the *u'* adornment in either base-view, no template for the join sequence is derived because the mediator cannot perform the join operation. Local join is ruled out because the attribute is not returned in the base-view query result, while bind join is ruled out because the attribute cannot be specified in the base-view query. Also, the *b* adornment of the second base view is treated as the *f* adornment only when the first base view query returns the attribute.

	f	**o**[S_{21}]	**b**	**c**[S_{22}]	**u**	**f'**	**o'**[S_{23}]	**b'**	**c'**[S_{24}]	**u'**
f	f	f	f	c[S_{22}]	f	f	o[S_{23}]	f	c[S_{24}]	-
o[S_{11}]	f	f	f	c[S_{22}]	f	f	c[S_{23}]	f	c[S_{24}]	-
b	b	b	b	c[S_{22}]	b	b	c[S_{23}]	b	c[S_{24}]	-
c[S_{12}]	c[S_{12}]	c[S_{12}]	c[S_{12}]	c[$S_{12} \cap S_{22}$]	c[S_{12}]	c[S_{12}]	c[$S_{12} \cap S_{23}$]	c[S_{12}]	c[$S_{12} \cap S_{24}$]	-
u	f	f	b	c[S_{22}]	f	f	o[S_{23}]	f	c[S_{24}]	-
f'	b	b	b	c[S_{22}]	b	b'	c'[S_{23}]	b'	c'[S_{24}]	-
o'[S_{13}]	c[S_{13}]	c[S_{13}]	c[S_{13}]	c[$S_{13} \cap S_{22}$]	c[S_{13}]	c'[S_{13}]	c'[$S_{13} \cap S_{23}$]	c'[S_{13}]	c'[$S_{13} \cap S_{24}$]	-
b'	b	b	b	c[S_{22}]	b	b'	c'[S_{23}]	b'	c'[S_{24}]	-
c'[S_{14}]	c[S_{14}]	c[S_{14}]	c[S_{14}]	c[$S_{14} \cap S_{22}$]	c[S_{14}]	c'[S_{14}]	c'[$S_{14} \cap S_{23}$]	c'[S_{14}]	c'[$S_{14} \cap S_{24}$]	-
u'	-	-	-	-	-	-	-	-	-	-

Table 9: Mapping Function for Join Sequences with Postprocessing and Output Restrictions.

7.2.3 Selection and Projection Views

The new mapping function for computing selection-view templates in a postprocessing mediator is given by Table 10. The top five rows of this table are identical to Table 4 because the mediator can perform all necessary postprocessing on the attribute that is output in the results of the base-view query.

In the bottom five rows, if the attribute does not participate in the selection predicate, its adornment is identical to the base-view adornment because no postprocessing of the attribute is undertaken by the mediator. If the attribute participates in the selection predicate, we may not be able to derive a selection-view template because the mediator cannot evaluate the selection predicate involving the invisible attribute when postprocessing the results of the base-view query. There are cases in which the mediator can avoid the need to perform postprocessing steps on the invisible attribute, based on its ability to infer required values for the invisible attribute from the selection predicate. If the mediator can infer a specific value that an invisible attribute must satisfy, it can input this value in the query to the base-view. Then, the mediator need not postprocess the result of the base-view query to check for the satisfaction of the required value for the invisible attribute. Note that if the base-view template has the *u'* adornment for an invisible attribute that participates in the selection predicate, the mediator will not be able to process the selection predicate and so it will not have a corresponding selection-view template.

In the case of *f'* and *b'* adornments of the base-view template, if the mediator can infer a required value for the attribute, then it can use this value to bind the base-view query and also

Base-View Adornment	Selection-Attribute Adornment	Nonselection-Attribute Adornment
f	f	f
o[S_1]	f	f
b	f or b	b
c[S_2]	f or c[S_2]	c[S_2]
u	f	f
f'	f or f'	f'
o'[S_3]	f or o'[S_3]	o'[S_3]
b'	f or b'	b'
c'[S_4]	f or c'[S_4]	c'[S_4]
u'	-	u'

Table 10: Mapping Function for Selection Views with Postprocessing and Output Restrictions.

allow the selection-view query to specify any value for that attribute. If the selection-view query specifies a value different from the value inferred from the predicate, the mediator can simply answer the selection-view query by returning an empty result. If the selection-view query specifies the same value, it can be used to bind the base-view query. If the selection-view query does not specify any value, the mediator can bind the base-view query with the value inferred from the selection predicate. Accordingly, for these two base-view adornments, we have the *f* adornment for the selection-view. In the case of the *o'* and *c'* base-view adornments, if the inferred value is one of the menu constants, the selection-view can have the *f* adornment.

The new mapping function for a projection view is given in Table 11. Note that the entries in the top five rows are identical to the entries in Table 5 because the mediator can perform postprocessing on the projected attribute. For the entries in the bottom five rows, the mediator cannot perform any postprocessing. So, it merely copies over the adornment from the base-view template.

7.3 Template Computation for Dynamic Mediators

Here we present the new mapping functions for the dynamic mediators of Section 6. Note that the dynamic mediators use the same algorithm as in Section 5 for computing the conservative templates of all its views. They also use the same algorithm for computing the liberal templates of union views, selection views and projection views. Therefore, the new mapping functions for computing all the templates except the liberal templates of join views are the same as those of Section 7.2. Here we present the new mapping function for computing liberal templates of join views.

Base-View Adornment	Projection-View Adornment
f	f
o$[S_1]$	f
b	b
c$[S_2]$	c$[S_2]$
u	f
f'	f'
o'$[S_3]$	o'$[S_3]$
b'	b'
c'$[S_4]$	c'$[S_4]$
u'	u'

Table 11: Mapping Function for Projection Views with Postprocessing and Output Restrictions.

	f	**o**$[S_{21}]$	**b**	**c**$[S_{22}]$	**u**	**f'**	**o'**$[S_{23}]$	**b'**	**c'**$[S_{24}]$	**u'**
f	f	f	f	f	f	f	f	f	f	-
o$[S_{11}]$	f	f	f	f	f	f	f	f	f	-
b	b	b	b	c$[S_{22}]$	b	b	c$[S_{23}]$	b	c$[S_{24}]$	-
c$[S_{12}]$	c$[S_{12}]$	c$[S_{12}]$	c$[S_{12}]$	c$[S_{12}]$	c$[S_{12}]$	c$[S_{12}]$	c$[S_{12}]$	c$[S_{12}]$	c$[S_{12}]$	-
u	f	f	f	f	f	f	f	f	f	-
f'	b	b	b	c$[S_{22}]$	b	b'	c'$[S_{23}]$	b'	c'$[S_{24}]$	-
o'$[S_{13}]$	c$[S_{13}]$	c$[S_{13}]$	c$[S_{13}]$	c$[S_{13} \cap S_{22}]$	c$[S_{13}]$	c'$[S_{13}]$	c'$[S_{13} \cap S_{23}]$	c'$[S_{13}]$	c'$[S_{13} \cap S_{24}]$	-
b'	b	b	b	c$[S_{22}]$	b	b'	c'$[S_{23}]$	b'	c'$[S_{24}]$	-
c'$[S_{14}]$	c$[S_{14}]$	c$[S_{14}]$	c$[S_{14}]$	c$[S_{14} \cap S_{22}]$	c$[S_{14}]$	c'$[S_{14}]$	c'$[S_{14} \cap S_{23}]$	c'$[S_{14}]$	c'$[S_{14} \cap S_{24}]$	-
u'	-	-	-	-	-	-	-	-	-	-

Table 12: Mapping Function for Liberal Templates of Join Views with Output Restrictions.

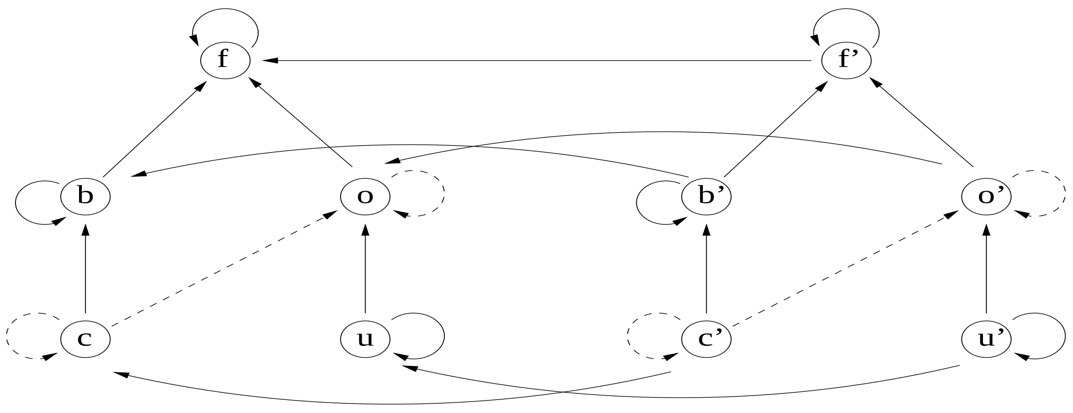

Figure 2: The New Adornment Graph.

The new mapping function to compute the liberal adornments of join views (Table 12) is quite similar to the one used in computing the conservative templates for join views with output restrictions (Table 9). The essential difference is that when we compute the liberal adornments, we optimistically assume that the bindings returned for the join attribute from the query on the first base view are in the menus of choices for the attribute when the second base-view adornments are *o*, *o'*, *c* and *c'*. For instance, when the first base-view adornment is *f* and the second base-view adornment is *c'*, Table 12 indicates the resulting adornment of *f* (instead of *c* in Table 9). We also have less constrained *c* and *o* adornments of the attribute in the join-view template when it has adornments with menus in both the base-view template. Note that when the first base-view does not return the join attribute (the five rows of the mapping function), the mapping function is identical to Table 9.

7.4 Minimization with New Adornments

The introduction of the new attribute adornments forces us to also reconsider the adornment graph of Figure 1, which is the basis for minimizing template sets. The new graph is given in Figure 2.

The restrictiveness of the five new adornments (*f'*, *o'*, *b'*, *c'* and *u'*) is compared in a manner similar to the comparison among the five adornments considered in the earlier adornment graph of Section 4. In addition, each of the five new adornments is more restrictive than its corresponding adornment that has no output restriction on the attribute. That is, *f'* is more restrictive than *f*, *b'* is more restrictive than *b*, and so on.

8 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.

We start by examining a mediator for two bookstores: Amazon.com [31] and BarnesAndNoble.com [35]. Amazon.com supplies the following three query forms [32,33,34]:

- Form 1: At least one of author, title and subject attributes must be specified.
- 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 (paperback or hardcover), price and shipinfo (availability and shipping information). We define the following view to describe the content of this web source:

$$Book_1(author, title, subject, KW, ISBN, pub, date, format, price, ship).$$

The query capabilities of Amazon.com are described by the templates in Table 13. The capabilities offered by each query form are captured by one or more templates. In Table 13, templates v_1, v_2 and v_3 capture Form 1, v_4 captures Form 2, while v_5, v_6 and v_7 capture Form 3.

	author	**title**	**subject**	**KW**	**ISBN**	**pub**	**date**	**format**	**price**	**ship**
v_1	b	f	f'	u'	u	u	u	u	u	u
v_2	f	b	f'	u'	u	u	u	u	u	u
v_3	f	f	b'	u'	u	u	u	u	u	u
v_4	u	u	u'	u'	b	u	u	u	u	u
v_5	u	u	u'	f'	u	b	f	u	u	u
v_6	u	u	u'	b'	u	f	f	u	u	u
v_7	u	u	u'	f'	u	f	b	u	u	u

Table 13: Templates of Amazon.com.

BarnesAndNoble.com [35] has two query forms. The first one supports search by ISBN. The second form allows search by specifying at least one of author, keywords and title attributes. In addition, through the second form, we can specify price limit, publishing format, subject area, and recommended age range. The output attribute set of BarnesAndNoble.com is the same as that of Amazon.com. We use the following schema for BarnesAndNoble.com.

$$Book_2(author, title, subject, KW, ISBN, pub, date, price, format, ship, age).$$

The capabilities of BarnesAndNoble.com are described by the templates in Table 14. Templates w_1, w_2 and w_3 describe the capabilities of the form through which we can search by author, keywords and title, while w_4 describes the form with the ISBN attribute. The price, format and subject attribute specifications have menus to choose from. These attributes can also be left unspecified. Therefore, they have the *o* adornment with their respective menus. For instance, the format attribute in w_1 has the *o* adornment with a menu of options: "Hardcover," "Paperback," "Books on Cassette or CD," and "Large Print." For the sake of simplicity, we do not list the menus of options for attribute adornments in the tables of this section.

	author	**title**	**subject**	**KW**	**ISBN**	**pub**	**date**	**format**	**price**	**ship**	**age**
w_1	f	b	o'	f'	u	u	u	o	o	u	o'
w_2	b	f	o'	f'	u	u	u	o	o	u	o'
w_3	f	f	o'	b'	u	u	u	o	o	u	o'
w_4	u	u	u'	u'	b	u	u	u	u	u	u'

Table 14: Templates of BarnesAndNoble.com.

Suppose we now build a mediator on these two Web bookstores. The mediator defines the following union view over the two source views:

$$Book(author, title, subject, KW, ISBN, pub, date, price, format, ship, age).$$

In order to compute the templates that describe the set of feasible queries on the union view, let us assume that the mediator employs postprocessing techniques. We ran our algorithm to compute mediator templates (see Section 7.2.1) and obtained 19 templates for the mediator view. These templates are shown in Table 15. The algorithm actually considered 28 (4×7) pairs of source-view templates and successfully generated union-view templates in the case of 23 pairs. However, the list of the resulting union templates has duplicates. After removing the duplicates, we ended up with 19 templates. We ran the minimization algorithm, based on the adornment graph of Figure 2, to obtain a concise set of 8 mediator templates (see Table 16).

From the minimized set of 8 templates for the mediator, 4 query forms can be derived. Corresponding to the first template in Table 16, the bookstore mediator has a query form that requires

	author	**title**	**subject**	**KW**	**ISBN**	**pub**	**date**	**format**	**price**	**ship**	**age**
t_1	b	b	o'	u'	f	f	f	f	f	f	u'
t_2	b	f	o'	u'	f	f	f	f	f	f	u'
t_3	b	f	u'	u'	b	f	f	f	f	f	u'
t_4	f	b	o'	u'	f	f	f	f	f	f	u'
t_5	f	b	u'	u'	b	f	f	f	f	f	u'
t_6	f	b	c'	u'	f	f	f	f	f	f	u'
t_7	b	f	c'	u'	f	f	f	f	f	f	u'
t_8	f	f	u'	u'	b	f	f	f	f	f	u'
t_9	f	b	u'	f'	f	b	f	f	f	f	u'
t_{10}	b	f	u'	f'	f	b	f	f	f	f	u'
t_{11}	f	f	u'	b'	f	b	f	f	f	f	u'
t_{12}	f	f	u'	u'	b	b	f	f	f	f	u'
t_{13}	f	b	u'	b'	f	f	f	f	f	f	u'
t_{14}	b	f	u'	b'	f	f	f	f	f	f	u'
t_{15}	f	f	u'	b'	f	f	f	f	f	f	u'
t_{16}	f	b	u'	f'	f	f	b	f	f	f	u'
t_{17}	b	f	u'	f'	f	f	b	f	f	f	u'
t_{18}	f	f	u'	b'	f	f	b	f	f	f	u'
t_{19}	f	f	u'	u'	b	f	b	f	f	f	u'

Table 15: Mediator Templates before Minimization.

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

Table 16: Mediator Templates after Minimization.

the specification of the ISBN attribute. Corresponding to the second template, we create a query form that requires the keywords to be specified. Next, we create 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 are 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 is 28 (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.

Our case study indicates that the capability-description languages used by many contemporary mediation systems are not suitable to describe sources on the Web. In particular, the lack of proper support for menus of constants and output restrictions in these languages is a big problem. In contrast, our capability-description framework (see Section 3) is well suited to describe the capabilities of Web sources and mediators.

We studied how query forms can be translated into templates and vice versa. In some cases, a query form can be described by a single template (e.g., template v_4 in Table 13 describes Form 2 of Amazon.com). In some other cases, a query form may be represented by a set of templates. For example, Form 1 of Amazon.com is described by templates v_1, v_2 and v_3 in Table 13. The most common situation where we need a set of templates to describe a form arises when the form specifies that at least one of a set of attributes must be filled in. We also studied the process of deriving query forms from templates. It is very easy to derive a query for for each template. For instance, in our bookstore mediator we could generate 8 query forms, one for each template. However, there are opportunities to capture the capabilities described by a set of templates through a single query form. For instance, we derived a query form for the bookstore mediator that covered four templates. The process of deriving query forms from templates involves steps that retrace the process of deriving templates from query forms. In general, we note that it is usually easier to derive templates from query forms than vice versa.

In our case study, we encountered interesting ways in which pop up menus are used in many Web forms. Some menus have a "default" option of "all" or "any." For example, the menu for the subject attribute in the search form of BarnesAndNoble.com has a default constant "all areas." In our capability-description framework, we can capture such an attribute restriction with either a c adornment or an o adornment. In the former case, we can list all the menu options, including the default one, in the constant set. If we use the o adornment, we can leave out the default option and gather all the other options. We adopted the latter approach in our case study.

In the course of our experiments, we noticed that many web sources change their query forms frequently. In our experience, sources change their Web forms many times in a short period of time (a few times a year). In this context, building mediators on such evolving sources poses special challenges. In particular, manual generation of query forms for mediators on Web sources becomes very difficult because whenever sources change, mediator capabilities have to be re-assessed. Automatic computation of mediator capabilities based on the techniques presented in this paper can be very helpful to mediation systems involving frequently changing sources.

We also observed that there are many Web sources that provide overlapping and related information. For example, there are many travel agencies on the Web. Mediators on the Web have to expect that large numbers of sources are integrated by mediator views. Once again systematic computation of mediator capabilities in this situation is essential. The process of minimizing template sets to arrive at concise capability description for mediators is also very important when mediators integrate large number of sources.

9 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 in which a rich set of attribute adornments are used 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 mediator capabilities for the various classes. 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. We also demonstrated that the techniques we developed in this paper yield concise capability description.

There are many promising avenues of related future work. First of all, we note that we used a simple definition of redundancy when computing minimal sets of templates. We considered a template to be redundant only when it is subsumed by another template. Although this test of redundancy covers a large number of situations, more opportunities for minimizing template sets accrue from a more general test of redundancy. In future, we plan to use a test based on subsuming a template with a set of templates, to arrive at smaller sets of templates for mediator views.

A second avenue of future work is the study of content-based mediators. Although we briefly touched upon the issue of using content information to enhance query answerability, a thorough treatment of how the capabilities of content-based mediators are computed is beyond the scope of this paper.

Another promising area of related future work is the consideration of *predicate capabilities* of sources and mediators. Sources can support sets of predicates on the data they export, and they can specify such predicates in their capability description [8]. A mediator can support predicates by translating the predicates in a user query into the supported predicates of the source queries. We need to develop machinery to compute the predicate capabilities of mediators based on those of its sources.

Finally, we note that in this paper we have not considered the computation of templates for recursive views. There are two situations in which recursive views occur. The first is when the definition of a view at a mediator is recursive (e.g., at mediator m_1, V is defined over W and W is defined over V). Another is when the definition of a view across mediators is recursive (e.g., at mediator m_1, V is defined over W of mediator m_2, and at mediator m_2, W is defined over V of mediator m_1). We have not dealt with either case of recursive views in this paper.

References

[1] S. Abiteboul, S. Cluet, T. Milo. Correspondence and Translation for Heterogeneous Data. *Proc. ICDT*, 1997.

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

[3] P. Buneman, S. Davidson, A. Kosky. Theoretical Aspects of Schema Merging. *Proc. EDBT Conference*, 1992.

[4] M. Carey, et al. Towards heterogeneous multimedia information systems: the Garlic approach. *RIDE Workshop*, 1995.

[5] S. Chawathe, et al. The TSIMMIS Project: Integration of Heterogeneous Information Sources. *IPSJ*, 1994.

[6] D. Chimenti, et al. An Overview of the LDL System. *Data Engineering Bulletin*, 10(4):52-62, 1987.

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

[8] J. Hammer, et al. Template-based Wrappers in the TSIMMIS System. *Proc. SIGMOD Conference* , 1997.

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

[10] R. King, M. Novak, C. Och, F. Velez. Sybil: Supporting Heterogeneous Database Interoperability with Lightweight Alliances. *The 3rd International Workshop on Next Generation Information Technologies and Systems, Israel*, 1997.

[11] T. Kirk, A. Levy, Y. Sagiv, D. Srivastava. The Information Manifold. *AAAI Symposium on Information Gathering in Distributed Heterogeneous Environment*, 1995.

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

[13] A. Levy, A. Rajaraman, J. Ordille. Query Answering Algorithms for Information Agents. *Proc. AAAI Conference*, 1996.

[14] A. Levy, M. Rousset. CARIN: A Representation Language Integrating Horn Rules and Description Logics. *AT&T Technical Report*, 1995.

[15] A. Levy, Y. Sagiv. Constraints and Redundancy in Datalog. *Proc. PODS Conference*, 1992.

[16] C. Li, et al. Capability Based Mediation in TSIMMIS. *Proc. SIGMOD Conference*, 1998.

[17] D. Maluf, G. Wiederhold. Abstraction of Representation for Interoperation. *Lecture Notes in AI, subseries of LNCS*, Vol. 1315, 1997.

[18] G. Mecca, et al. The Araneus Web-Based Management System. *Proc. SIGMOD Conference*, 1998.

[19] R. Miller, Y. Ioannidis, R. Ramakrishnan. Schema Equivalence in Heterogeneous Systems: Bridging Theory and Practice. *Information Systems*, 19:3-31, 1994.

[20] K. Morris, J. Ullman, A. Gelder. Design Overview of the NAIL! System. *Proc. ICLP*, 1986.

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

[22] Y. Papakonstantinou, H. Garcia-Molina, J. Widom. Object Exchange across Heterogeneous Information Sources. *Proc. ICDE Conference*, 251–260, 1995.

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

[24] M. Roth, P. Schwarz. Don't Scrap It, Wrap it! A Wrapper Architecture for Legacy Data Sources. *Proc. VLDB Conference*, 1997.

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

[26] S. Tsur, C. Zaniolo. LDL: A Logic-Based Data Language. *Proc. VLDB Conference*, 1986.

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

[28] R. Yerneni, C. Li, H. Garcia-Molina, J. Ullman. Computing Capabilities of Mediators – Extended Version. `http://www-db.stanford.edu/pub/papers/medtemp.ps`.

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

[30] A1Books.com Query Forms. `http://www.a1books.com/cgi-bin/a1books/search.cgi?act=advSearch/`

[31] Amazon.com Web Site. `http://www.amazon.com/`

[32] Author-Title-Subject Query Form for Amazon.com. `http://www.amazon.com/exec/obidos/ats-query-page/`

[33] ISBN Query Form for Amazon.com. `http://www.amazon.com/exec/obidos/subst/search/isbn.html/`

[34] Keywords-Publisher-Date Query Form for Amazon.com. `http://www.amazon.com/exec/obidos/subst/search/publication-date.html/`

[35] BarnesAndNoble.com Query Forms. `http://shop.barnesandnoble.com/booksearch/search.asp?`

[36] Extensible Markup Language (XML) 1.0: Proposed Recommendation. `http://www.w3.org/TR/REC-xml`

[37] Yahoo Shopping Guide for Books. `http://shopguide.yahoo.com/shopguide/books.html`