

# **IDB: Toward the Scalable Integration of Queryable Internet Data Sources**

Jaewoo Kang

Mong Li Lee

Jeffrey F. Naughton

University of Wisconsin-Madison  
Computer Sciences Department  
1210 West Dayton Street  
Madison, WI 53706

{jaewoo, leeml, naughton}@cs.wisc.edu

## **Abstract**

As the number of databases accessible on the Web grows, the ability to execute queries spanning multiple heterogeneous queryable sources is becoming increasingly important. To date, research in this area has focused on providing semantic completeness, and has generated solutions that work well when querying over a relatively small number of databases that have static and well-defined schemas. Unfortunately, these solutions do not extend to the scale of the present Internet, let alone the Internet of the future. In this paper, we present an approach that makes the opposite tradeoff: it provides a scalable, unified view over large numbers of queryable information sources by sacrificing some expressive power in the set of queries supported. We have developed a prototype system, IDB, which implements this approach. The IDB system provides scalability through three main techniques. First, it uses a collection of ontologies organized into hierarchical namespaces as a medium for expressing data semantics. Second, it employs a declarative query language to describe information sources so that source descriptions can be "executed" at run time instead of being pre-compiled into the system. Third, it utilizes inverted-index style operations to identify the subset of information sources that are relevant to a particular user query. We describe the design, architecture, and implementation of IDB and illustrate its use through case examples.

## **1. Introduction**

As the number of information sources on the web continues to grow, the need for good information integration technology increases. For this reason, the problem of information integration has received a great deal of attention from the research community [CGH+94, PGG+95, PGW95, P96, GPQ+97, DG97, GDK97, KLS+95, LRO96a, LRO96b, FPN+99] and from the commercial community [Mys, Pri, Cad, Bot]. The differences between the approaches taken by the research community and the commercial community are striking. At the risk of oversimplifying the issue, the research community has focused on the semantic power of the integration technology at the expense of the scalability of the approach, while the commercial

community has focused on scalability at the expense of semantic power. In this paper we describe IDB, a system that attempts to find a middle ground between these two extremes.

Research systems such as TSIMMIS [CGH+94, PGG+95, PGW95, P96, GPQ+97], the Information Manifold [KLS+95, LRO96a, LRO96b], and Infomaster [DG97, GKD97] provide a general query facility over the integrated view of a number of data sources. They are able to support powerful queries and infer implicit joins through the use of a view containment test. While this provides the very real advantage that a user can get an answer to a query without knowing that a join is required to construct this answer (since the system deduces the implicit join “under the covers”), unfortunately, it also renders query planning and execution expensive. Query planning and execution in such systems is expensive because the size of the plan space that must be searched and generated query plan grows quickly with the number of sources wrapped [LRO96b, P96, UI197]. For this reason, these systems are most effective at evaluating complex queries over a relatively small number of sites.

In the commercial world, the information integration space is dominated by comparison shopping services [Mys, Pri, Cad, Bot]. In contrast to the research systems, these systems do not provide general purpose querying – their goal is to be able to evaluate a small number of canned queries (expressed by forms presented to the user) over a large number of sites.

Another very real barrier to scalability that is largely orthogonal to the semantic power of an integration system is the difficulty of “wrapping” new sites, that is, how hard it is to add new information sources to the system. The research community has not really dealt with this scalability issue (there is little call to wrap hundreds of sites in a research prototype!), while the commercial community “solves” this problem by employing a small army of programmers to write wrappers, usually aided by some sort of wrapper generation toolkit.

In this paper, we will demonstrate how IDB addresses these two issues by employing namespaces in its query facility and “soft-wrapping” information sources. Figure 1 illustrates the main components of the IDB system. Unlike the commercial services, IDB aims to provide a general query facility by using a collection of ontologies organized into a hierarchical *namespace*. Each ontology in the IDB namespace defines a set of terms that describes common concepts. This namespace is used as the medium for expressing data semantics. Both user queries and source descriptions are written using terms in the namespace.

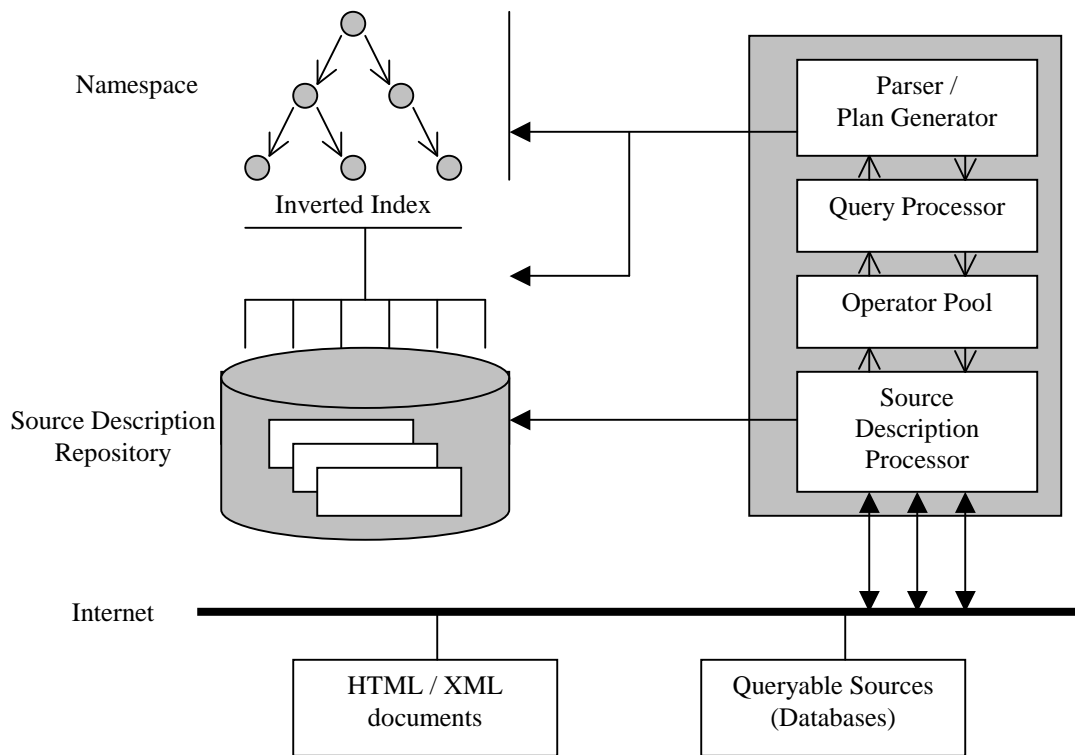
The IDB query language, IDBQL, is based on SQL. IDBQL queries are expressed using terms from the IDB namespace. When writing a query, users do not need to know about the exported views of each individual information source. Instead, the query engine will identify a set of relevant information sources by using terms that appear in the query to probe an inverted index.

Unlike TSIMMIS and the Information Manifold, IDB does not infer any implicit joins. Clearly, this means that IDB can answer only a subset of the queries handled by these systems. However, this also implies that query planning in IDB is trivial (it requires simple inverted list lookup operations) and scales to large numbers of sites. Whether this is the right tradeoff or not will depend upon the application in question – we suspect there is a need for both kinds of systems.

We also propose a novel approach called *soft-wrapping* to wrap information sources. In IDB, a “wrapper” is just a declarative query evaluated at runtime. Source descriptions can be

“executed” at run time instead of being pre-compiled into the system (or “hard-wrapped”). The advantages of soft-wrapping over hard-wrapping are many. First, it is more flexible and portable, because the writing of source descriptions is independent of any run-time environment. Second, soft-wrappers can be tested and registered dynamically at runtime through a Web interface, without having to restart the system. Third, it is easy to adapt to dynamically changing Web data sources, as recompilation is not needed. Finally, soft wrapping is more secure in that what is registered is a declarative query, and not a pre-compiled wrapper program that must be trusted by whoever executes the wrapper.

In Section 2, we describe our namespace approach. Section 3 elaborates on the soft wrapper approach. IDBQL and query examples are described in Section 4. Section 5 explains the query evaluation in IDB. Finally, we present our conclusion in Section 6.



**Figure 1. IDB Prototype System Architecture**

## 2. Namespaces

IDB uses a collection of ontologies organized into hierarchical namespaces as a medium for expressing data semantics. An IDB ontology is a grouping of terms describing a concept. The terms in the ontologies are fully reusable. When defining an ontology, one can borrow existing terms from other ontologies in the namespace as well as create new terms. An ontology can selectively inherit (or reuse) any subset of the parent ontology. Inheritance from multiple ontologies is also allowed.

The IDB namespace functions as a global schema that provides a uniform view over information on the Web. It is an *a priori* schema as opposed to the *a posteriori* schema of TSIMMIS [PGG+95, PGW95, P96, GPQ+97]. In TSIMMIS, user queries are formulated over the view exported by a mediator. The mediated view is, in turn, generated by integrating views of lower level mediators or data sources. As a result, any source level changes such as adding a new source or dropping an existing source may affect the upper level mediated view user queries being formulated on. On the other hand, the IDB namespace is defined independently from the views of data sources. In fact, the source view is defined using the terms in the namespace. Because of this, information source level changes do not affect the global view.

The current implementation of IDB uses a simple collection of terms as the global schema. In future, if XML namespaces become prevalent, they could be used in place of the IDB ontology [NAM99, SCH99, RDF98, XML98]. By adopting XML namespaces, we would then be able to reuse a large number of widely used namespaces as our schema without having to reinvent them.

Figure 2 illustrates some instances of IDB namespaces. The movie ontology consists of terms that may be useful to describe movies. The term `product#name` from the product namespace is reused in the movie namespace as `movie#title`. It is advantageous to reuse existing terms, as this increases the number of sources that can contribute to a given query. For instance, if user queries on the name of a product using the product ontology, then information sources belonging to book and movie ontologies are also queried in addition to sources directly belonging to the product ontology. This is because the `book#title` and `movie#title` terms are inherited from the `product#name` term in the product namespace.

NAMESPACE <code>book#</code>		NAMESPACE <code>movie#</code>	
ATTRIBUTES		ATTRIBUTES	
<code>vendor</code>	<code>product#vendor</code>	<code>vendor</code>	<code>product#vendor</code>
<code>category</code>	<code>product#category</code>	<code>category</code>	<code>product#category</code>
<code>title</code>	<code>product#name</code>	<code>title</code>	<code>product#name</code>
<code>url</code>	<code>product#url</code>	<code>url</code>	<code>product#url</code>
<code>author</code>		<code>director</code>	
<code>year</code>	<code>product#year</code>	<code>actor</code>	
<code>price</code>	<code>product#price</code>	<code>year</code>	<code>product#year</code>
<code>format</code>		<code>price</code>	<code>product#price</code>
<code>stock</code>		<code>format</code>	
<code>publisher</code>		<code>stock</code>	
<code>isbn</code>		<code>studio</code>	<code>product#make</code>
		<code>note</code>	<code>product#note</code>
		<code>genre</code>	
		<code>rating</code>	

**Figure 2. Namespaces for Book and Movie Ontology**

### 3. Source Description Language

IDB interacts with information sources using source descriptions. The role of source descriptions in the IDB approach is twofold:

- They export the views and capabilities of information sources.
- They extract and map local data in the described source to the exported view of the source.

Unlike traditional “hard wrapping”, the IDB approach uses a “soft wrapping” scheme that allows source descriptions to be executed at query evaluation time. The IDB’s source description is, in fact, a query language that “queries” a remote document or database. As a result, IDB does not require hard-coded or compiled wrappers to communicate with sources. On the other hand, previous approaches, including those used in the Information Manifold, TSIMMIS, and Infomaster, use “hard wrappers” that require recompilation each time an information source changes its data presentation.

The syntax of the IDB source description language is as follows:

```
SELECT  list-of-terms
FROM    url [post|get] [html|xml]
WHERE   mapping-rule [[and|or] mapping-rule] ...
```

The SELECT clause defines the exported view, the FROM clause specifies the location of the remote database and its query capability while the WHERE clause defines the mapping rules. Figure 3 shows a source description for **amazon.com**. After evaluating the source description of **amazon.com**, an eight-column table of vendor, title, etc. will be generated.

The execution starts by evaluating the FROM clause. The FROM clause specifies the location of **amazon.com**’s book database and the query binding that it accepts. **Amazon.com**’s book database is published on the Web through a front-end form interface. This form interface accepts user inputs on the title and author fields, and this information is encoded in the url string in the FROM clause. In the case where the target information source is a document, the url of the document can simply be placed in the FROM clause without any query binding encoding.

Once IDB has rewritten the user query into local queries, the placeholders \$book#title\$ and \$book#author\$ will be replaced with the corresponding values from the user query. After it opens a url connection and sends the query string, the query result will be returned from the source in an HTML page. The HTML page is parsed into a DOM tree [DOM98]. If a source returns an XML page, then IDB will invoke an XML parser instead to generate the DOM tree. After this parsing step, the remaining query processing steps are transparent to both XML and HTML since the DOM interface is generic to both markup languages.

The WHERE clause consists of a set of path expressions and perl-style text operations. The path expressions are evaluated over the DOM tree generated from the result page. The syntax of our path expression is like that of HEL [SA99a, SA99b] and WIDL [All97]. HEL also supports perl-style pattern matching. The IDB source description language, however, allows direct mapping from path expressions to the exported view and provides a larger set of text operations. Further, it allows the conjunction and disjunction of path expressions. For instance, depending on

the user query binding, the amazon.com database returns two different types of HTML pages. In case the user query binding results in exactly one book entry, it directly returns the HTML page that contains the full book description. Otherwise, it returns an HTML page that contains a list of matching book entries, where each book entry has a short description and url to the book page. We need different path expressions for each of these cases, as shown in Figure 3.

```

SELECT  'Amazon' AS book#vendor, book#title+, book#url, book#author,
        book#year, book#price+, book#format, book#stock
FROM    http://www.amazon.com/ats-query?author=$book#author&title=
        $book#title$ post
WHERE
    .html.body.table[1].tr->dl(
        .dt[*](
            .b.a.getAttribute(href):$book#url;
            .b.a.txt():$book#title;
            .nobr.font.txt():$book#stock;
        );
        .dd[*](
            .pcdata[0](
                .txt():$book#author,          /([^\/]*)\//m;
                .txt():$book#format,         /([^\/]*)\//([^\/]*)\//m;
                .txt():$book#year,           /Published\D*(\d+)/im;
            )
            .pcdata[1].txt():$book#price,    /Price\D*([\d\.]+)/im;
        );
    );
OR
.html.body(
    .table[1].tr.td[2].form(
        .strong.font.txt():$book#title;
        .font.a[*].txt():$book#author,     j/, /;
    );
    .font.txt():$book#price,                /Our Price:\s*\$([\d\.]+)/im;
    .font.p.pdata[0].txt():$book#stock;
    .font[1].b.txt():$book#format;
    .font[1].txt(1):$book#year,            /(\d+)\)/m;
)

```

**Figure 3. Source Description for the Amazon.com Book Database<sup>1</sup>**

<sup>1</sup> The URL in the FROM clause is shortened for illustration.

The IDB source description language supports popular perl regular expression operations [WCS96] including **match**, **substitute**, **join**, **split**, and a custom-designed **switch** operator. The switch operator is used to normalize the irregularity of output data across multiple sources. For example, some sources represent product availability in graphical symbols and they must be transformed into the text equivalents. The dot(.) in the path expression implies the direct path from the parent element to the child and the arrow(->) implies 0 or more steps exist in between.

The SELECT clause provides a global semantics for the local data. It defines the schema of the table that is generated by the execution of the source description. Note that the constant value ‘amazon’ is materialized into the book#vendor term as all book entries are coming from the same source, amazon.com. The plus(+) sign at the end of an attribute is shorthand for ‘IS NOT NULL’.

Although the exported view of Figure 3 consists of terms only from the book ontology, this is not a requirement of the IDB approach. The IDB source description can choose terms selectively from one or more ontologies. The source description need not conform to any namespace nor have any restriction on choosing sets of terms from various ontologies. This allows the IDB source description language to describe sources using terms that are close as possible to the original semantics of the data.

As is the case in amazon.com, data extracted from the result page can potentially have some nested structure. To map this nested data to a flat output table, IDB employs a set of special **iterators** that are associated with each output attribute.

As we pointed out earlier, since IDB uses a declarative query language for its source descriptions, the traditional pre-compiled “wrapper” is no longer needed. This “soft wrapper” approach is more scalable since the process of writing, testing, and registering wrappers is not dependent on the hardware and software development environment, thus it can be completely decentralized. In fact, the source descriptions can be tested and registered at runtime through the Internet without bringing down the system. Anyone can write and register the source descriptions from anywhere in the Internet. Also, with the soft wrapper approach it is easier to adjust to dynamically changing Web sources, as they need not be recompiled each time the source changes. Finally, it is more secure in that what is registered is a declarative query and not a pre-compiled wrapper program. That is, using the soft-wrapper approach, a “buggy” wrapper may cause the data from the wrapped source to be mapped incorrectly, but since it is just a declarative query, it does not pose a security risk to the site executing the soft wrapper query.

#### 4. The IDB Query Language

In this section we introduce the IDB query language, IDBQL, primarily through examples. IDBQL is a subset of SQL with additional keyword predicates. The keyword predicates are added to support a keyword match operation that is perhaps the most popular operation in real-world web queries.

A query is formulated using the terms in the IDB ontology. The query writer does not need to know about the exported views of each of the individual underlying information sources. The query processor will identify a set of relevant information sources by probing an inverted index using the terms used in the query. We will describe the details of query processing in the next section.

Keyword Operators	Semantics
$A \sim B$	True, if all keywords in B appear in A.
$A =\sim B$	True, if all keywords in A appear in B.
$A \sim\sim B$	True, if either $A \sim B$ or $A =\sim B$ is true.

**Table 1. Keyword Operator Semantics**

Our first example query illustrates a basic structure of IDBQL and the use of the keyword predicate.

```
SELECT    B.vendor, B.title, B.author, B.price, B.year
FROM      book B
WHERE     B.title  $\sim$  'Database Systems'
```

This query uses the “book” ontology and retrieves vendor, title, author, price, and year information of books whose title contains the keyword ‘Database’ and ‘Systems’. The result table will include book entries with titles, e.g. ‘Database Management Systems’, ‘Readings in Database Systems’, etc. The partial output of the query is shown below.

```
vendor: Bookpool
title: Database Management Systems
author: Raghu Ramakrishnan, et al
price: 55.95
year: 1999

vendor: Amazon
title: Fundamentals of Database Systems
author: Ramez A. Elmasri/ Shamkant B. Navathe
price: 79.75
year: 1999

vendor: Barnes and Noble
title: A First Course in Database Systems
author: Jeffrey D. Ullman, With Jennifer Widom
price: 59.75
year: 1997
```

The keyword operators are especially useful since the data that the IDB is dealing with comes from autonomous Web information sources. The presentation format of the data may differ across the Web sites, and perhaps even the data within one site may have different presentation formats over time. One common example would be the format of person’s name. Some sources may put the last name before the first name, and some others first name first. IDB supports three keyword operators and their semantics is defined in Table 1.

Literal Operand	Attribute (String)
String	No coercion. If predicate operator is '=' evaluate string equality, otherwise fail.
Integer	Parse the string into Real. If parsing succeeds coerce the Integer operand to Real and evaluate predicate, otherwise fails.
Real	Parse the string into Real. If parsing succeeds evaluate predicate, otherwise fails.

**Table 2. Data Type Coercion Rules**

The first example query was not very selective as returned more than 200 entries from various online book vendors. Our second example query adds two more selection conditions to the first query and retrieves book availability information along with the original attributes.

```
SELECT B.vendor, B.title, B.author, B.price, B.year, B.stock
FROM   book B
WHERE  B.title ~= 'Database Systems' AND
       B.author ~= 'Ramakrishnan' AND
       B.year > 1998
```

This query illustrates the use of numeric order predicates and data type coercion. The IDB data model is essentially type free. All attribute values are treated as string literals. To evaluate an order predicate (<, >, >=, <=, =), IDB coerces the attribute value using the rules shown in Table 2. For instance, in the above query, if the year attribute is not null and can be parsed into a number, the predicate will be evaluated over two numeric values. In a join predicate, such as book.year = movie.year, both operands are attributes. In this case, one of the attributes is first coerced into an appropriate type before the predicate is evaluated using the rules in Table 2.

Part of the result table for the second query is shown below.

vendor:	Fatbrain
title:	Database Management Systems , Second Edition
author:	Ramakrishnan , Raghu / Gehrke, Johannes
price:	53.25
year:	1999
stock:	Ships same day
vendor:	Bookpool
title:	Database Management Systems
author:	Raghu Ramakrishnan, et al
price:	55.95
year:	1999

stock:	In Stock!
vendor:	Amazon
title:	Database Management Systems
author:	Raghu Ramakrishnan, Johannes Gehrke
price:	75.60
year:	1999
stock:	Usually ships in 24 hours

The last example query is a simple join query. It is provided to illustrate the case where more than one ontology is involved in a query. It retrieves title, actor of movies and vendor, url, format, price of books where the movies are directed by 'Steven Spielberg', books are written by 'Michael Crichton', and both movie and book have the same title.

```
SELECT M.title, M.actor, B.vendor, B.url, B.format, B.price
FROM   book B, movie M
WHERE  B.author ~= 'Michael Crichton' AND
       M.director ~= 'Steven Spielberg' AND
       B.title = M.title
```

Part of the result table for this query is shown below.

title:	Jurassic Park
actor:	Morgan Freeman / Nigel Hawthorne / Anthony Hopkins / Sir Anthony Hopkines
vendor:	Amazon
url:	<a href="http://www.amazon.com/exec/obidos/ASIN/0394588169/">http://www.amazon.com/exec/obidos/ASIN/0394588169/ ...</a>
format:	Hardcover
price:	18.87
title:	Jurassic Park
actor:	Morgan Freeman / Nigel Hawthorne / Anthony Hopkins / Sir Anthony Hopkines
vendor:	Barnes and Noble
url:	<a href="http://shop.barnesandnoble.com/booksearch/isbnInquiry.asp?isbn=0345370775..">http://shop.barnesandnoble.com/booksearch/isbnInquiry.asp?isbn=0345370775..</a>
format:	Mass Market Paperback
price:	6.39
title:	Jurassic Park
actor:	Morgan Freeman / Nigel Hawthorne / Anthony Hopkins / Sir Anthony Hopkines
vendor:	Borders
url:	<a href="http://search.borders.com/cgi-bin/db2www/search/search.d2w/Details?prodID...">http://search.borders.com/cgi-bin/db2www/search/search.d2w/Details?prodID...</a>
format:	Paperback
price:	6.39

## 5. Query Processing

The following are the steps of IDB query processing.

- A user query is formulated using terms in the IDB ontology.
- The query engine identifies base tables for each namespace used in the query. A base table is determined by identifying the minimum subset of terms in a given namespace that are required to evaluate the user query.
- The query engine retrieves a set of source descriptions for each base table from the source description index. This index is, in fact, an inverted index that associates terms in the IDB ontology to the relevant source descriptions. We will illustrate this in detail later in this section.
- The query engine reformulates the original user query using the views exported from the set of source descriptions that were identified in the previous step.
- The query engine materializes local views at each source, unions results by base tables, and processes remaining predicates (e.g. joins between base tables).

We illustrate these steps using a query example. The following are the two ontologies that our example query references. We only show the list of terms for illustration.

book (vendor, title, url, author, year, price, format, stock, publisher, isbn)  
review (isbn, title, author, year, review)

The example query is shown below. It retrieves the vendor, title, price, and review attributes of books that have the keywords ‘Database’ and ‘Systems’ in their title.

```
SELECT    B.vendor, B.title, B.author, B.price, R.review
FROM      book B, review R
WHERE     B.title ~= 'Database Systems' and B.title = R.title
```

The first step of the query processing is to identify the base tables and the predicate binding implied for the base tables. To illustrate this process, we represent the above query in rules with adornments.

```
queryfbfff(vendor, title, author, price, review) :-
    bookfbff(vendor, title, author, price),
    reviewbf(title, review),
    title ~= 'Database Systems'
```

Predicate adornment is used to illustrate how the binding pattern serves as a filter for pruning out irrelevant sources. As shown above, the title is the only variable that is bound in this query. The base tables in this query are book<sup>fbff</sup>(vendor, title, author, price) and review<sup>bf</sup>(title, review). The way base tables are identified is straightforward. We

gather all terms used in the SELECT and WHERE clause and group them into the namespace that appears in the FROM clause.

Note that the base table is different from the global predicates used in the Information Manifold [LRO96a, LRO96b]. Unlike the predefined set of predicates in the Information Manifold, the base table of IDB is dynamically generated by projecting out terms from the particular user query. Also, compared to other systems, IDB uses a much simpler way to identify information sources that are relevant to a particular user query. Previous systems, including the Information Manifold, TSIMMIS [PGG+95, PGW95, P96, GPQ+97], and Infomaster [DG97, GKD97] identify information sources through a query rewriting scheme based on the view containment test in their query processing (see [UII97] for an overview). In contrast, IDB utilizes inverted index style operations in query processing. To illustrate this, let us assume we have the following information sources.

```
amazonfbff (vendor, title, author, year, price)
bordersfbff (vendor, title, author, price)
book1fbff (vendor, title, price, isbn)
book2fb (author, isbn)

nytimesbff (title, author, review)
wpostbbf (title, isbn, review)
```

Due to the space limitation, we only show the list of terms that each source exports instead of showing source descriptions. For the actual example of source description see the amazon.com's source description in Figure 3.

The adornment here has a slightly different meaning than the adornment in the query. It is used here to specify the query capability. For instance, the adornment of *amazon*, *fbff*, means that *amazon* can accept the queries on either title or author and returns a table of columns including vendor, title, author, year, and price. Similarly, *nytimes* can answer the queries on title and returns a table of title, author, year, and review.

When a source description is registered, IDB indexes the source into two inverted indexes. The first inverted index, *A*, maintains the relation between all terms used in the source description and the identifier of the source description itself. The second index, *B*, indexes only bound variables (terms). A snap shot of the indexes are shown in the Figure 4.

For each base table in the query, IDB identifies a subset of source descriptions using the indexes. In our example query, we first probe the index *A* using all terms in the book base table that include vendor, title, author, and price. We get *amazon* and *borders* by intersecting the four resulting inverted lists. Second, we probe the index *B* using bound variables in the book base table. Here, title is the only bound variable in the user query. It returns *amazon*, *borders*, and *book1*. Finally, we intersect the two results to get the subset of sources that are relevant to the user query, specifically for the book base table. We repeat the same process to get *nytimes* and *wpost* for the review base table.

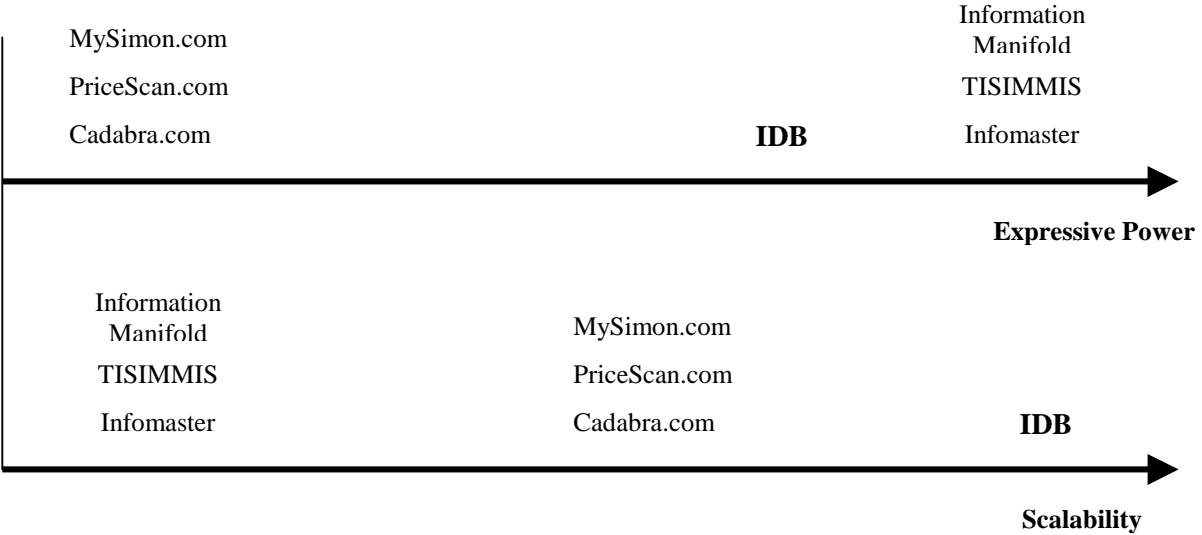
Inverted Index A:	
book#vendor	-> amazon, borders, book1
book#title	-> amazon, borders, book1
book#author	-> amazon, borders, book2
book#year	-> amazon
book#price	-> amazon, borders, book1
book#isbn	-> book1, book2
review#title	-> nytimes, wpost
review#author	-> nytimes
review#isbn	-> wpost
review#review	-> nytimes, wpost
Inverted Index B:	
book#title	-> amazon, borders, book1
book#author	-> amazon
book#isbn	-> book2
review#title	-> nytimes, wpost
review#isbn	-> wpost

**Figure 4. Snapshot of Inverted Index after Registering the Sample Sources**

All sources in the result have the query capability on the title attribute and can produce a table of the projected columns in each base table. The remaining steps of the query processing are straightforward. We group the source descriptions on the base tables and execute them. When executing source descriptions, the placeholders in the FROM clause are replaced by the query binding and encoded into a legal url query string. In our example, the source descriptions of `amazon` and `borders` are executed and generate a book table by unioning the results from both sources. Similarly, the source descriptions of `nytimes` and `wpost` are executed to generate the review table.

Earlier integration systems based on the view containment test may find more results than IDB. For instance, the Information Manifold and TSIMMIS would generate tuples from the sources `book1` and `book2` by joining on the attribute `isbn`. Because IDB does not do any inference, it would not find this implicit join. As we have said in the introduction, IDB is less semantically powerful than preceding systems. By giving up on this power, IDB gains flexibility and scalability. Whether this tradeoff is the correct one or not will depend upon the application using the system.

While implicit joining is powerful and useful in many cases, in other instances its omission may not significantly impact the utility of the system. Implicit joins may not always be practical in Web integration for three reasons. First, to utilize implicit joins, sources may have to be accessed sequentially. For instance, in our example the source `book1` must be queried before `book2` can be queried as the required input to `book2`, `isbn`, is retrieved from the result of `book1`. Second, depending on the selectivity of user predicates, the number of times a local



**Figure 5. Expressive Power and Scalability Characteristics of Information Integration Systems.**

query needs to be executed may increase exponentially. For example, if `book1` returns a list of 10 matching books, `book2` need to be queried 10 times for each distinct isbn number from the query result of `book1`. In this example, `book2` will return exactly one match. But if `book2` also returns 10 results each time and there is a third source, say `book3`, whose input binds to the result of `book2`, now `book3` has to be queried 100 times. Third, implicit joins may not always yield correct results if, because of the autonomous nature of Web data, two equivalent objects have different presentations or two different objects have the same presentation.

The last step of query processing is to evaluate join predicates across the namespaces. In our example, the book and review tables are joined on title attribute.

## 6. Conclusion

We have implemented a prototype version of the IDB and have built a simple application over book and movie domains. To date, we have wrapped 14 book sources and 15 movie sources. Wrapping a new source takes 10 minutes on average. This application runs IDBQL queries (including joins) over the union of the sources. There is little difference in performance between running the application over 2 sources versus running the application over all 29. The query planning stage is instantaneous, with all of the delay in query evaluation due to waiting for the information sources to respond, and due to the multithreading of the IDB query engine, the delay in waiting for the sites to respond is overlapped. Although this small application is clearly not sufficient to demonstrate scalability, our experience in building the application was encouraging. In the future we plan to use the IDB to build more complex applications over an order of magnitude more information sources.

The IDB approach represents a compromise between the two extremes of commercial comparison shopping systems and research information integration prototypes. Figure 5 illustrates the expressive power and scalability characteristics of these systems. We have put the IDB to the right of the comparison shopping services in the scalability dimension because to the

best of our knowledge these services use the compiled hard-wrapper approach. Clearly, there is room for all three types of systems; each is appropriate for different kinds of applications. The commercial comparison shopping services are good for applications characterized by a small number of canned queries over a large number of sites, whereas the systems favored by the research community are good for applications that require powerful queries with implicit joins over a small number of sites. The IDB approach, through its combination of soft wrapping, namespaces with simple mappings to source locations, and query semantics requiring explicit joins, supports a general query language and promises to scale to a large number of sites.

## References

[All97] Charles Allen. WIDL: Application Integration with XML. *World Wide Web Journal*, 2[4], November 1997

[Bot] <http://www.bottomdollar.com>

[Cad] <http://www.cadabra.com>

[CGH+94] Sudarshan S. Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey D. Ullman, and Jennifer Widom. The tsimmis project: Integration of heterogeneous information sources. In *Proc. Information Processing Society of Japan Conference*, pages 7-18, 1994.

[DG97] Oliver M. Duschka and Michael R. Genesereth. Answering Recursive Queries Using Views. In *Proc. Principles of Database Systems*, 1997.

[DOM98] Document Object Model (DOM) Level 1 Specification. <http://www.w3.org/TR/REC-DOM-Level-1/>

[FPN+99] Jerry Fowler, Brad Perry, Marian Nodine, and Bruce Bargmeyer. Agent-Based Semantic Interoperability in InfoSleuth. In *proceedings of ACM SIGMOD Conference*, 1999.

[GKD97] Michael R. Genesereth, Arthur M. Keller, and Oliver Duschka, Infomaster: An Information Integration System. In *proceedings of 1997 ACM SIGMOD Conference*, May 1997.

[GPQ+97] Hector Garcia-Molina, Yannis Papakonstantinou, Dallon Quass, Anand Rajaraman, Yehoshua Sagiv, Jeffrey D. Ullman, Vasilis Vassalos, and Jennifer Widom. The tsimmis approach to mediation: Data models and languages. *Journal of Intelligent Information Systems*, 8(1):117-132, 1997.

[KLS+95] T. Kirk, Alon Y. Levy, Y. Sagiv, and D. Srivastava. The Information Manifold. *AAAI Spring Symp. On Information Gathering*, 1995.

[LRO96a] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Query-answering algorithms for information agents. In *Proc. of the AAAI Thirteenth National Conference on Artificial Intelligence*, pages 40-47, 1996.

**[LRO96b]** Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. Int. Conf. on Very Large Data Bases*, pages 251-262, 1996.

**[Mys]** <http://www.mysimon.com>

**[NAM99]** Tim Bray, Dave Hollander, and Andrew Layman. Namespaces in xml. <http://www.w3.org/TR/REC-xml-names>, 1999.

**[P96]** Y. Papakonstantinou. Query processing in heterogeneous information sources. PhD thesis, Dept. of Computer Science, Stanford University, 1996.

**[PGG+95]** Yannis Papakonstantinou, A. Gupta, Hector Garcia-Molina, and Jeffrey D. Ullman. A Query Translation Scheme for Rapid Implementation of Wrappers. *Fourth DOOD*, Singapore, December, 1995.

**[PGW95]** Yannis Papakonstantinou, Hector Garcia-Molina, and Jennifer Widom. Object Exchange across Heterogeneous Information Sources. *Intl. Conf. On DataEngineering*. March, 1995.

**[Pri]** <http://www.pricescan.com>

**[RDF98]** Dan Brickley, R.V. Guha, and Andrew Layman. Resource description framework (rdf) schema specification. <http://www.w3.org/TR/WD-rdf-schema>, 1998.

**[SA99a]** Arnaud Sahuguet and Fabien Azavant. Looking at the Web through XML glasses. In *Proc. CoopIs*, 1999

**[SA99b]** Arnaud Sahuguet and Fabien Azavant. Web Ecology: Recycling HTML pages as XML documents using W4F. WebDB, 1999

**[SCH99]** Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. Xml schema. <http://www.w3.org/TR/xmlschema-1/>, 1999.

**[Ull97]** Jeffrey D. Ullman. Information integration using logical views. In *Proc. Int. Conf. on Database Theory*, pages 19-40, 1997.

**[WCS96]** Larry Wall, Tom Christiansen, and Randal L. Schwartz. Programming Perl. O'Reilly, 1996.

**[XML98]** Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. Extensible markup language (xml) 1.0. <http://www.w3.org/TR/REC-xml>, 1998.