General Strategy for Querying Web Sources in a Data Federation Environment

IGI PUBLISHING
701 E. Chocolate Avenue, Hershey PA 17033-1240, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.igi-global.com

ITJ 4842

This paper appears in the publication, Journal of Database Management, Volume 20, Issue 2
edited by **Keng Siau © 2009, IGI Global**

General Strategy for Querying Web Sources in a Data Federation Environment

Aykut Firat, Northeastern University, USA

Lynn Wu, Massachusetts Institute of Technology, USA

Stuart Madnick, Massachusetts Institute of Technology, USA

ABSTRACT

Modern database management systems are supporting the inclusion and querying of non-relational sources within a data federation environment via wrappers. Wrapper development for Web sources, however, is a convolution of code with extraction and query planning knowledge and becomes a daunting task. We use IBM DB2 federation engine to demonstrate the challenges of incorporating Web sources into a data federation. We, then, present a practical and general strategy for the inclusion and querying of Web sources without requiring any changes in the underlying data federation technology. This strategy separates the code and knowledge in wrapper development by introducing a general-purpose capabilities-aware mini query-planner and a data extraction engine. As a result, Web sources can be included in a data federation system faster, and maintained easier. [Article copies are available for purchase from InfoSci-on-Demand.com]

Keywords: *Capability Restriction; Federated Database; Wrapper; Query Planning*

INTRODUCTION

Federated databases offer information integration on demand in dynamic environments, where data warehousing approaches are not feasible (Sheth & Larson, 1990; Geer, 2003). In modern relational database management systems, even non-relational sources can be included in a data federation via "wrappers" so that they can be queried as if they are part of a single large database (Somani, Choy, & Kleewein, 2002; Thiran, Hainaut, Houben, & Benslimane, 2006). Wrappers are mechanisms by which the federated server interacts with non-relational data sources by performing operations such as connecting to a data source and retrieving data from it iteratively.

Retrieving data from Web sources, however, is complicated because data is semistructured and Web sources may have requirements (e.g., they may require forms to be filled before returning data); thus general-purpose wrappers

for arbitrary Web pages are not provided in data federation systems. Instead the user needs to implement a custom wrapper for each Web source by coding data extraction patterns and parts of the federated query planning protocol in a low-level programming language such as C. This convolution of code with the data extraction and planning knowledge turns wrapper development into a daunting task, results in code duplication, and slows down the data federation process.

Within the last decade or so, many research projects (Papakonstantinou, Gupta, & Haas, 1998; Levy, Rajaraman, & Ordille, 1996; Li & Chang, 2000; Zadorozhny, Bright, Vidal, Raschid, & Urhan, 2002; Li, 2003; Pentaris & Ioannidis, 2006) offered algorithmic solutions to "query planning with source restrictions." The goal of these studies was to offer an expressive language to specify source restrictions, and let the federated query planner come up with an optimal plan using this knowledge. These approaches do not need any cooperation from the individual data sources other than knowing about their limitations. Had they found their way into commercial systems, they would eliminate part of the code and knowledge convolution problem: the wrapper developer would only need to code the data extraction knowledge and not worry about the query planning aspects. Yet the separation of code and knowledge would still not be satisfactorily achieved in non-cooperative federated query planners. For this study, we have chosen to work with IBM DB2's cooperative federated query planner, which poses more challenges than the non-cooperative ones. Our focus is on improving the usability and maintenance aspects of the wrapper development process without requiring any changes in its underlying data federation technology. We do not offer yet another proposal to rewrite a state-of-the-art distributed query planner (Kossmann, 2000), or create an independent infrastructure for querying Internet data sources (Braumandl et al., 2001; Suciu, 2002), but provide a non-intrusive approach that works with what is available today with minimal effort.

We have tested our prototype implementation with numerous Web sites. A moderate user with no programming experience can include a typical Web site into a data federation in less than an hour. The process often takes much longer when the existing procedural coding approach is used by an experienced programmer. Furthermore, explaining, learning, and tutoring wrapper development becomes much easier, as the task changes from writing and debugging a *program* to specifying and debugging *knowledge*.

In the rest of this paper, we start with a motivational example that illustrates the need for data federation involving Web sources. We then provide some background on data federation with non-relational data sources and describe the current architectural difficulties of incorporating a Web source. Next, we describe our approach to wrapper development, and the algorithms used to perform planning and optimization for Web sources with capability restrictions. We end with an overview of related work and future research issues.

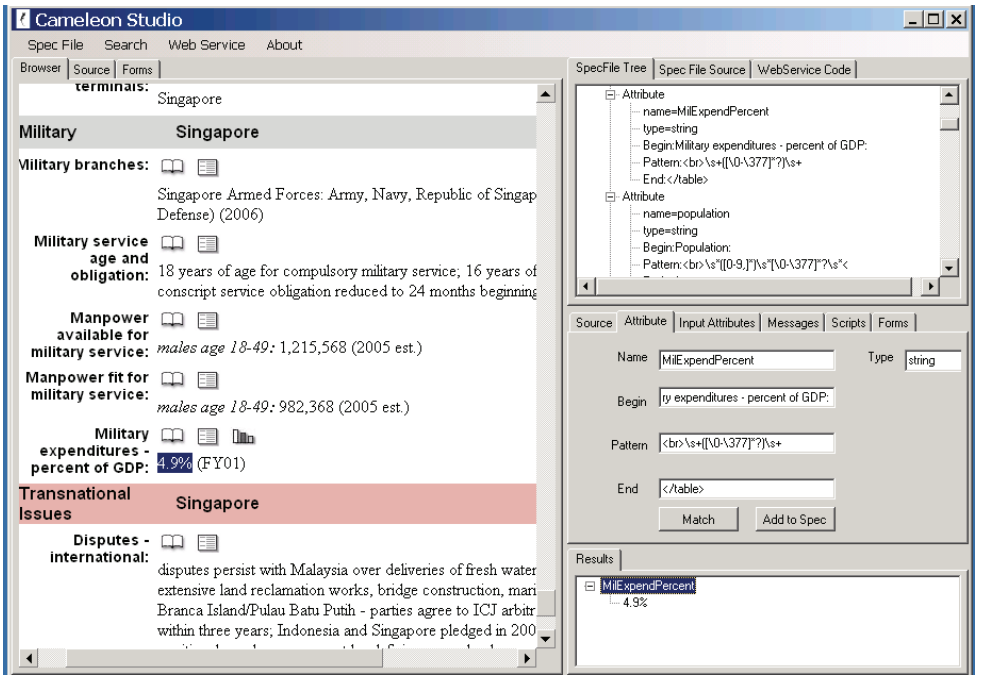
MOTIVATIONAL EXAMPLE

Consider, first, finding the *military expenditure per capita* of countries in the world using the CIA world fact book Web site. This information is scattered inside the world fact book (see Figure 1), and first needs to be located and extracted. By using the Web wrapper, Cameleon# (Firat, Madnick, Yahaya, Kuan, & Bressan, 2005; Firat, Madnick, & Siegel, 2000) and its accompanied visual helper, Cameleon# Studio, we can wrap the CIA world fact book site using simple regular expressions and treat it as a very simple relational table as illustrated in Figures 2 and 3.

The Cameleon# wrapper engine's main functionality is, however, extraction and thus is only able to answer SQL queries involving a single source, with required inputs bound to a single set of values at a time. For that reason, we decided to use the powerful query planning, optimizing, and execution capabili-

Figure 1. Available data in CIA World Fact Book site

Figure 2. CIA World Fact Book site is visually wrapped with Cameleon# Wrapper Engine



ties of a commercial data federation engine to handle more complex query situations. Using the extended architecture to be described later on, we define a nickname for our Web source in DB2 as shown below:

```
CREATE NICKNAME CIA (
country char(20),
population dec(10,1),
GDP dec(10, 2),
GDP_unit char(20),
MilExpendPercent dec(9,4) for server Cameleon#_server
options(SERVER_NAME 'http://interchange.mit.edu/Cameleon_sharp/camserv.aspx?',
PREDICATES 'country')
```

The options in the above description indicate the location of Cameleon# information extraction server, and the required input column *country*. We are then able to treat the CIA fact book like a relational table and issue the following query using DB2:

```
Q1: SELECT country, population,
   GDP, gdp_unit, MilExpendPer-
   cent
       FROM cia
       WHERE country IN
```

("Singapore", "Israel", "United States", "United Kingdom", "Malaysia")

(see Table 1)

Since we want to calculate the *military expenditure per capita*, we need to perform the appropriate calculation with a mathematical expression. In addition, we must perform unit conversions (e.g., adjust for the fact that some GDP values are in billions and some in trillions) with the auxiliary database table scalefactor:

TEXT	SCALE
Billion	1000000000
Trillion	1000000000000

This is achieved by joining the non-relational CIA Web source with the relational scalefactor table using the following query:

```
Q2: SELECT country, (MilExpendPercent *
      GDP * scalefactor.scale / population)
AS MilExpPerCapita
   FROM cia, scalefactor
  WHERE scalefactor.text=cia.gdp_unit AND
country IN
```

("Singapore", "Israel", "United States", "United Kingdom", "Malaysia")

COUNTRY	MilExpPerCapita
Singapore	1379.85
Israel	1901.93
United States	1674.64
United Kingdom	716.72
Malaysia	238.91

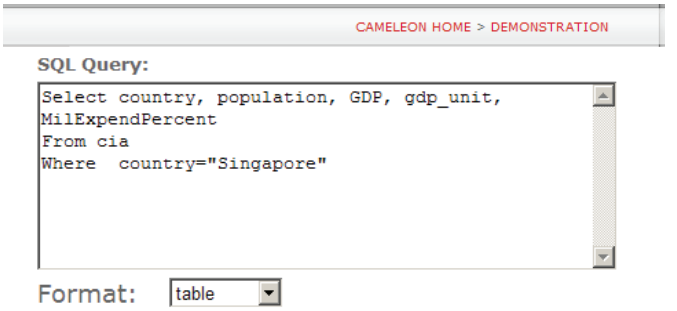
Finally, we would like to obtain the *military expenditure per soldier* by creating another NICKNAME for a Wikipedia Web source that has the sizes of armed forces and formulating a federated query joining multiple Web sources, as shown in Figure 4.

As this simple example shows, querying Web sources using a data federation offers many operational benefits. One can take advantage of the relational database technology in processing semistructured Web data. For example, Web sources can be joined with each other and with other sources, calculations and set operations can be performed, and queries can be optimized. Currently, however, even setting up this motivational example is extremely difficult, if not impossible using one of the data federation

Table 1.

COUNTRY	POPULATION	GDP	GDP_UNIT	MILEXPEND-PERCENT
Singapore	4492150	126.5	billion	4.90
Israel	6352117	156.9	billion	7.70
United States	298444215	12.31	trillion	4.06
United Kingdom	60609153	1.81	trillion	2.40
Malaysia	24385858	287	billion	2.03

Figure 3. Simple SQL Query against the wrapped CIA World Fact Book



country	population	gdp	gdp_unit	milexpendpercent
Singapore	4492150	126.5	billion	4.90

Figure 4. Available data in Wikipedia


```
CREATE NICKNAME ARMFORCES (
country char(15),
armed_forces integer)
for server Cameleon#_server
options(SERVER_NAME
'http://interchange.mit.edu/Cameleon#_sharp
/camserv.aspx?',
PREDICATES 'country')
```

```
Q3: SELECT cia.country, armed_forces,
      (MilExpendPercent*GDP*scalefactor.uni
      t)/ (armed_forces*1000) AS
      milpersoldier
FROM cia, armforces, scalefactor
WHERE cia.country IN ('Singapore',
'Israel',
```

COUNTRY	*ARMED_FORCES*	*MILPERSOLDIER*
Singapore	*60*	*103308.33*
Israel	*168*	*71912.50*
United States	*1426*	*350481.07*
United Kingdom	*190*	*228631.58*
Malaysia	*110*	*52964.54*

engines. The most direct solution offered by DB2 requires coding a custom wrapper for each Web source, but even then those Web sources cannot be joined with each other on the required input attributes (IBM, 2006).

We designed and implemented a new architecture that drastically accelerates the inclusion and querying of Web sources in a data federation. The motivational example, for instance, can be set up in less than an hour without any low-level programming. Users only need to locate and specify the information they want to use on the Web with Cameleon# Studio--a point and click helper tool--and define the Web sources with data definition statements similar to classical "CREATE TABLE" statements. Before explaining the details of our extended architecture, we provide background on the typical operation of data federation systems by using DB2 and its Request-Reply-Compensate protocol as an example.

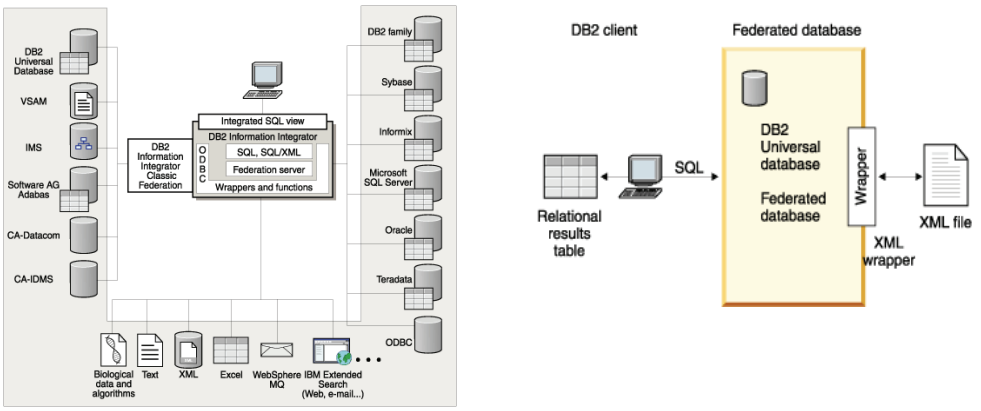
QUERYING NON-RELATIONAL SOURCES IN A DATA FEDERATION

The goal of a data federation system is to allow clients to access diverse and distributed data sources, regardless of location, format, or access language, from a single interface. While data federation may have a slower access performance compared to data consolidation (as in data warehousing), it has the benefits of (i) *reduced implementation and maintenance costs*, (ii) *access to current data from the source of record*, and (iii) *combining traditional data with mixed format data* (IBM, 2006; Haas, Lin, & Roth, 2002).

As shown in Figure 5, a data federation system uses wrappers to access non-relational data sources such as flat files, XML pages, and Web services. After the user submits a query, the federated server collaborates with the wrapper for each data source to generate an optimized access plan for the query and then evaluates it. Such a plan might call for parts of the query to be processed by the wrappers, by the federated server, or partly by the wrappers and partly by the federated server. The federated server chooses among the plans primarily on the basis of cost.

Upon receiving the *request*, the wrapper indicates which sub-pieces of the query fragment it can evaluate, and puts this information in the *reply* to the request. Request properties such as cost, cardinality and ordering properties can also be included. For a typical request, a wrapper

Figure 5. IBM DB2 data federation architecture [Adopted from (DB2 Information Center 2006)]



could return zero or more reply objects. Each reply represents a different accepted fragment. By the end of query planning, the federated server will weigh all the cost estimations and determine a query execution plan incorporating some set of the accepted fragments offered up by the wrapper in response to requests. During query execution, the federated server will ask the wrapper to execute these query fragments. The federated server can also *compensate* for any query fragments that have not been accepted. Examples of this include a complex predicate or sorting that is beyond the capability of the data source in question. This protocol is therefore called a *request-reply-compensate protocol* in IBM DB2.

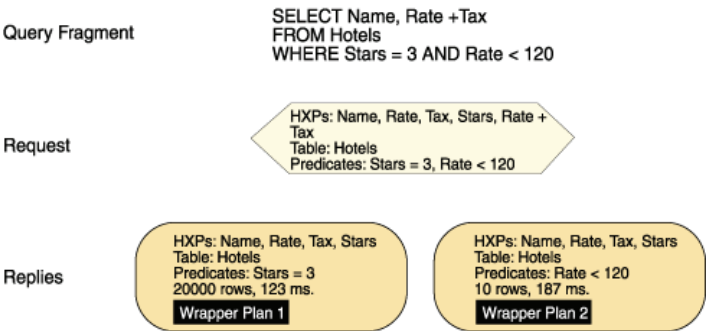
Consider Figure 6 as an example. The query fragment (`SELECT Name, Rate + Tax FROM Hotels WHERE Stars=3 AND Rate < 120`) is passed to the wrapper as a request by indicating the head expressions (HXPs), table name, and the predicates. In this case, we assume that the wrapper cannot handle the complete request as it cannot do the `Rate + Tax` calculation and it cannot do two predicates at a time, so replies with two separate parts, which when combined in the federated server answers the original query.

The request-reply-compensate protocol offers a generic framework allowing the federated server to communicate with non-relational data sources through a black box wrapper. Among the built in wrappers that comes with IBM DB2, there are two that are particularly relevant to querying Web sources: XML and Web services wrappers. These wrappers can be used if Web sources can be turned into XML format, or Web services. Neither of these, however, satisfies our desire to include an arbitrary Web source in a data federation and query them without artificial restrictions. The XML wrapper, for instance, does not have the concept of a required input attribute: the XML page should be accessible with a fixed address. Many Web sources are dynamically generated based on input attributes, which precludes the use of XML wrapper as it is. The Web services wrapper, on the other hand, has artificial query restrictions such as "*no IN or OR predicates are allowed for input columns*" (IBM, 2006). For instance, even our simplest query `Q1` cannot be handled by the Web services wrapper assuming our Web source was somehow turned into a Web service.

THREE-TIER ARCHITECTURE FOR QUERYING WEB SOURCES IN A DATA FEDERATION

The solution we offer for the inclusion and querying of Web sources in a data federation involves extending the existing two-tier custom

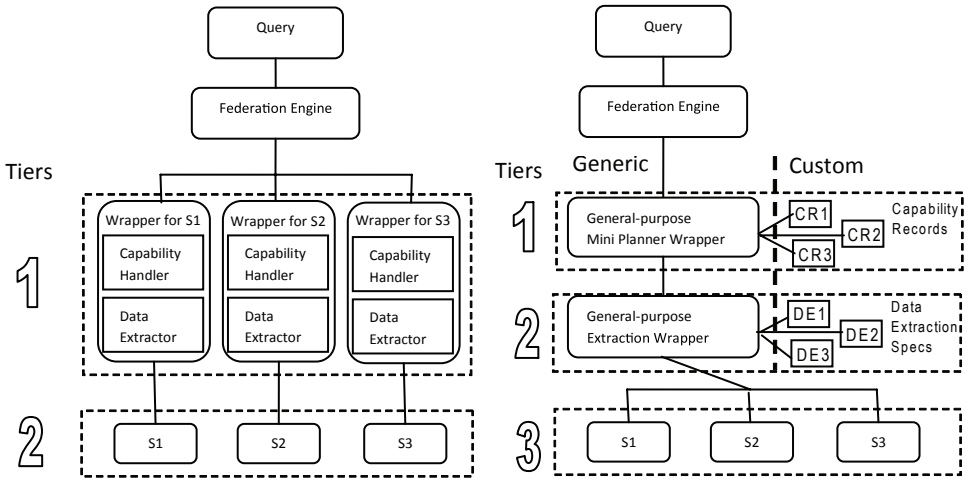
Figure 6. Request-Reply-Compensate protocol example [Adopted from (DB2 Information Center 2006)]



wrapper architecture into a three-tier architecture while separating the generic and custom aspects of wrapper development as shown in Figure 7. This new architecture separates code and knowledge, minimizes redundancy, and complements the central query planner when incorporating web sources in a data federation by following the wrapper development protocol specified in DB2 Information Integrator Wrapper Developer's Guide (IBM, 2004).

In the first tier of our solution we have a general-purpose mini planner-wrapper responsible for planning queries involving Web sources. We call it a mini-planner because Web sources have characteristics that limit the query planning space; therefore we do not have to deal with the complexity of a traditional query planner. Our mini planner, in most cases, only needs to handle query planning for a single Web source, leaving complex planning involving multiple sources to the federated server. The mini planner can run Web source queries in parallel, and order them intelligently when they are joined, while respecting their capabilities. Web source capabilities are expressed using simple capability records, which indicate the required input attributes, and whether input attributes can be bound to more than one value at a time. For example, the capability record [b(1), f, f, f, f] for the CIA nickname means that the first attribute *country* needs to be bound with one value at a time (b(1)), and the rest of the attributes must be free (f). In general b(N) indicates that the attribute can be bound with up to N values; "f" indicates that the attribute must be free; and "?" indicates that the attribute can be either bound or free. These capability records are implemented for each source using the nickname definition, right after the predicate keyword. The second-tier is a general-purpose data extraction engine responsible for retrieving data from a Web source and presenting it in the format expected by the data federation engine. For this task, any capable general-purpose data extraction engine can be used. We used the data extraction engine, Cameleon#, which uses declarative rules based on regular expressions to extract data from Web pages. Cameleon#

Figure 7. Comparison of Architectures. The extended architecture separates data extraction and capability handling functionalities. Furthermore the primary wrapper is responsible for planning queries posed against web sources with capability restrictions



a. Existing two-tier architecture *b. Extended three-tier architecture*

Studio can be used to help generate the necessary specification file. An example specification file is shown in Figure 8.

The second-tier extraction wrapper accepts these specification files as input to extract data from any Web source without any procedural coding. Next, we provide the details of the mini query planner for Web sources with capability restrictions.

MINI QUERY PLANNER FOR WEB SOURCES

The mini query planner creates a plan that can efficiently retrieve remote data while satisfying query restrictions. Generally, a query planning engine needs to decompose the original query into component subqueries (CSQ), such that each CSQ can be answered using a single data source (Alatovic, 2001; Fynn, 1997). Our mini query planning engine does not need to perform the decomposition since the federated database engine already divides the original query into CSQs, known as requests, where each request can be processed by a single data source. In addition to query decomposition, a query planning engine also needs to maintain the CSQ execution order. Typically, independent CSQs are executed first, followed by dependent CSQs that can be answered using prior results. Thus, detecting the dependencies among the CSQs is crucial to successful planning. Our query planning engine uses both the federated engine and capability records to analyze CSQ dependencies. When the CSQ dependency can be determined using query semantics, our query planning engine uses the federated database engine. When a CSQ does not meet all the capability restrictions of a source, however, the query planning engine will determine if information from other parts of the query can be used to satisfy the capability restrictions. If the restrictions can be satisfied, the CSQ will be modified with the required information so that it can be answered by the native data source.

The simplest case for a query execution plan (QEP) is when all CSQs meet the capability restrictions imposed by their native data sources, and they can be executed independently and in parallel. In this case, the federated engine simply decomposes the original query into CSQs and sends them to the native sources through wrappers. After receiving all processed row sets from the native sources, the federated engine aggregates the data and returns the final result.

When a CSQ cannot be executed by itself, however, it is necessary to determine if the CSQ can still be processed using results from other CSQs. Two procedures are used to determine the dependencies: the first method relies on detecting dependencies using query semantics; the second method employs the capability records to meet any unsatisfied restrictions using information from other processed CSQs. The next two sections describe in detail how the two procedures work and how they compensate for each other.

Dependencies Detected via Query Semantics

In Figure 9, we show an example dependency between CSQs that can be detected using the query semantics. In this example, the original query is decomposed into an inner-select CSQ, which can be executed independently, and an outer-select CSQ, which depends on the data returned by the inner-select CSQ. The federated engine facilitates the detection of this dependency by tagging country attribute with a type called "*unbound kind"* to signal to the wrapper that the binding values would be available after the inner-select CSQ is executed. Once the result from the inner-select CSQ is returned, the wrapper needs to create a new set of CSQs by replacing the *"unbound kind"* tag in the original CSQ with the returned value(s). In this example, as illustrated in Figure 9, since country names are returned from the inner-select CSQ; (e.g., Albania, Andorra, Austria, Belarus, etc.), new CSQs are formed after binding each country name to the country attribute. The wrapper then needs to send this new set of CSQs to the native data source. Once the native source processes

Figure 8. Example Specification File For the CIA Web Source

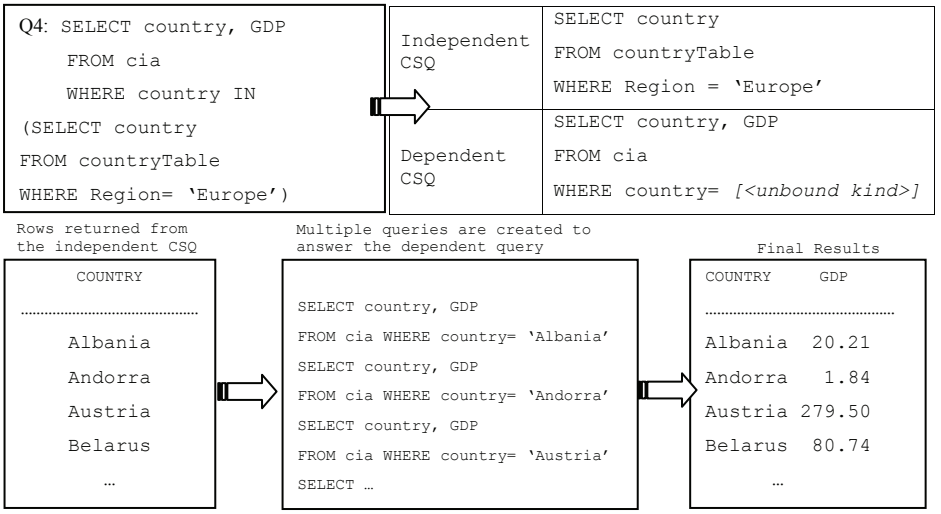
```
<?xml version="1.0" encoding="UTF-8" ?>
- <RELATION name="cia">
  - <SOURCE URI="https://www.cia.gov/cia/publications/factbook/index.html">
    - <ATTRIBUTE name="Link" type="string">
      - <BEGIN>
          <![CDATA[ <body ]]>
        </BEGIN>
      - <PATTERN>
          <![CDATA[ <option value="([^"]*)"[^>]*>#Country# ]]>
        </PATTERN>
      - <END>
          <![CDATA[ </[Bb][oO][dD][yY]> ]]>
        </END>
      </ATTRIBUTE>
    </SOURCE>
  - <SOURCE URI="https://www.cia.gov/cia/publications/factbook/#Link#">
    - <ATTRIBUTE name="MilExpendPercent" type="string">
      - <BEGIN>
          <![CDATA[ Military expenditures - percent of GDP: ]]>
        </BEGIN>
      - <PATTERN>
          <![CDATA[ <br>\s+([\0-\377]*?)\s+ ]]>
        </PATTERN>
      - <END>
          <![CDATA[ </table> ]]>
        </END>
      </ATTRIBUTE>
    - <ATTRIBUTE name="population" type="string">
      - <BEGIN>
          <![CDATA[ Population: ]]>
        </BEGIN>
      - <PATTERN>
          <![CDATA[ <br>\s*([0-9,]*)\s*[\0-\377]*?\s*< ]]>
        </PATTERN>
      - <END>
          <![CDATA[ </tr> ]]>
        </END>
      </ATTRIBUTE>
    - <ATTRIBUTE name="GDP" type="string">
      - <BEGIN>
          <![CDATA[ purchasing\s*power\s*parity ]]>
        </BEGIN>
      - <PATTERN>
          <![CDATA[ <br>\s+[$]*([,0-9\.]*)[\0-\377]*?\s*</td> ]]>
        </PATTERN>
      - <END>
          <![CDATA[ </tr> ]]>
        </END>
      </ATTRIBUTE>
    - <ATTRIBUTE name="GDP_unit" type="string">
      - <BEGIN>
          <![CDATA[ purchasing\s*power\s*parity ]]>
        </BEGIN>
      - <PATTERN>
          <![CDATA[ <br>\s+[$]*[,0-9\.]*([\0-\377]*?)\s*\( ]]>
        </PATTERN>
      - <END>
          <![CDATA[ </tr> ]]>
        </END>
      </ATTRIBUTE>
    </SOURCE>
</RELATION>
```

the CSQs, the wrapper needs to assemble the results and return them to the federation engine. In this example, the wrapper sends the queries to the CIA Web source, retrieves the GDP values and returns them to the federated engine.

Dependencies Implied by Capability Restrictions

Some CSQ dependencies may be not be detected via query semantics, but are implied by capability restrictions. Consider for example query Q5, which asks for the GDP and armed-

Figure 9. An example query dependency that can be detected by query semantics



force size of countries that are ranked in the top 10 both in terms of highest GDP and largest armed-force size. Like in the previous example, *countryTable* is a relational source that has the list of countries and their regions.

```
Q5: SELECT cia.country, armed_
forces, GDP
FROM countryTable,
(SELECT GDP
 FROM cia
 ORDER BY GDP DESC FETCH FIRST
10 ROWS ONLY) cia,
(SELECT armed_forces
 FROM armforces
 ORDER BY armed_forces DESC FETCH
FIRST 10 ROWS ONLY) armforces
WHERE cia.country = country-
Table.country AND
armforces.country = country-
Table.country
```

To process this query, the query planning engine needs to invoke the countryTable relation to retrieve the list of all countries, and then pass them to the cia and armed_forces relations to obtain the requested data. In order to answer this query, however, the federated engine creates the following two CSQs on Web sources:

```
CSQ1: SELECT GDP
 FROM cia

CSQ2: SELECT armed_forces
 FROM armforces
```

Since none of the CSQs has unbound parameters, the federated engine assumes that they can be executed independently by using the native data sources. Both Web sources, however, require that *country* must be bound before they can return any results. Thus, we cannot produce an answer to the query by only using query semantics. If we consider the capability information, however, it is possible to process both CSQs by finding the missing information from other parts of the query. Using the join conditions "cia.country = countryTable.country" and "armforces.country = countryTable.country" we can rewrite CSQ1 and CSQ2 into CSQ3 and CSQ4 by providing the values for the country attribute from the countryTable relation:

```
CSQ3 : SELECT GDP
   FROM cia
   WHERE country IN
   (SELECT country FROM count-
ryTable)

CSQ4 : SELECT armed_forces
   FROM armforces
   WHERE country IN
   (SELECT country FROM count-
ryTable)
```

With this added condition, CSQ3 and CSQ4 satisfy the capability restrictions and thus can be processed by the native sources. Although CSQ3 depends on the result from countryTable, this dependency can now be resolved via query semantics with the help of the federation engine as in Figure 10.

The query execution plan (QEP) algorithm, which uses capability records to process CSQs, is presented in Figure 11. The algorithm is based on finding independently executable CSQs in the query and processing them before any dependent CSQs. In most cases, the CSQs that cannot be executed independently lack at least one binding restriction. Once such a CSQ is detected, the algorithm determines if the CSQ can still be executed by searching for the missing binding from other CSQs. If the algorithm finds the missing binding, it is incorporated into the CSQ so that it can be processed by the native source.

There are two non-trivial steps in this algorithm: a) determining if a CSQ can be independently executed (step 3), and b) deciding whether a CSQ can be processed using join bindings from a set of executed CSQs (step 8). The details of these two steps are illustrated in the following sections.

Figure 10. An example query dependency implied by capability restrictions

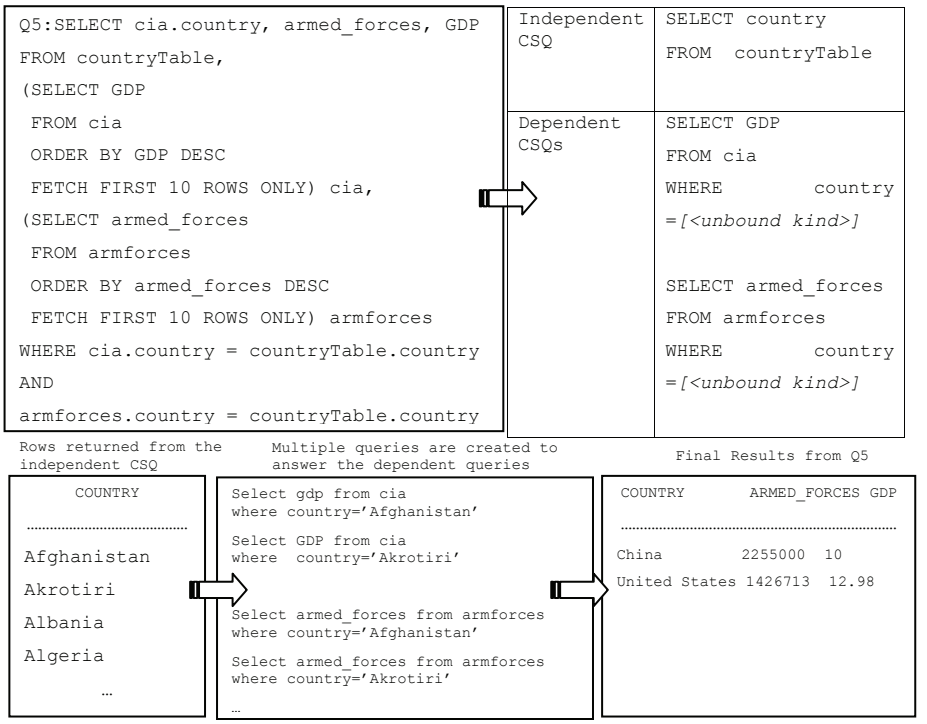


Figure 11. QEP generation algorithm supporting binding query restrictions

```
Input: Single Query q
Output: Query Execution Plan (QEP)

QEP Generation Algorithm:
1.   initialize set S to an empty set
2.   for all CSQs c in S
3.        if c is independently executable
4.        add c to set S
5.        add entry 0:c to QEP
6.   repeat until no more CSQs are added to S
7.        for all CSQs c outside of S
8.             if CSQ c can be executed using bindings from CSQs in S
9.               add an entry for c to QEP including all join bindings of c
10.              add CSQ c to set S
11.   if S does not contain all CSQs in a query
12.        throw exception "query cannot be executed"
13. return QEP
```

Determining Independently Executable CSQs

Figure 12 shows the algorithm for determining whether a CSQ is independently executable. The algorithm uses the capability restrictions to detect any missing binding in the CSQ, and if they exist, the algorithm determines if these binding conditions can still be satisfied.

Determining Whether a CSQ is Executable Given a set of Executed CSQs

The algorithm for determining whether a CSQ is executable, given a set of CSQs that have already been executed, is depicted in Figure 13. Consider the earlier example in Figure 10 once more. Although the cia and armforces CSQs cannot be executed independently, they can still be processed by finding the missing binding through the use of join conditions in the query. This algorithm detects this class of CSQs that are missing bindings, but can still be executed using information made available through executing other parts of the query. For the specific example of Figure 10, upon finding the attribute country to be unbound, the algorithm discovers a joint binding, countryTable.country=cia.country, that can provide the missing values to the attribute *country*. After modifying the cia CSQ with the new joint binding, the cia CSQ can be executed. Similarly, the binding for armforces CSQ is discovered from the countryTable.country=armforces.country predicate; the CSQ is modified and executed.

Handling Key-at-a-Time Query Restriction

Many Web sources require a single key value to be provided at a time. Consider for example the query Q1 again. (Cia web source has b(1) – one binding at a time – restriction on the attribute country):

Figure 12. Algorithm for determining whether a CSQ is independently executable

```
Independently Executable CSQ:
1. for all binding specifiers bs of c's underlying relation r
2.     for all attribute specifiers as of bs
3.     if as is of type bound and there is no binding in CSQ c
for corresponding attribute
4.                continue 1
5.           else
6.                continue 2
7.      end for
8.    return true
9.   end for
```

Figure 13. Algorithm for determining whether a CSQ is executable given a set of executed CSQs

```
Input:    set of executed CSQs S, new CSQ n
Output:  if n cannot be executed given join bindings from CSQs in S
                returns null
            else
                returns list of join bindings for CSQ n
CSQ Executable:
1.   for all binding specifiers bs of CSQ n
2.        initialize list of join bindings to an empty list jbl
3.        for all attribute specifiers as of bs
4.             if as is of type bound and CSQ n does not contain binding
                  for attribute matching as
5.                  if there is a join binding jb from n's attribute
                        matching as to one of CSQs in S
6.                  add jb to jbl
7.                  continue 3
8.            else
9.                  continue 1
10.       return jbl
11. return null
```

```
Q1: SELECT country, population,
GDP,  gdp_unit,  MilExpendPer-
cent
    FROM cia
    WHERE country IN
```

("Singapore", "Israel", "United States", "United Kingdom", "Malaysia")

In order to answer this query, the mini-planner needs to change the query into a union of four one-key-at-a-time queries, and perform

the union operations locally in parallel. In general, Web sources may have b(N) – N binding at a time – restriction. The short algorithm, shown in Figure 14, handles the general case by recursively rewriting the original query into subqueries. Finally, the algorithm returns the result by performing the union operator on the results of all the subqueries.

Cost Statistics Generation

Cost statistics are especially important for federated queries (Kache, Han, Markl, Raman, & Ewen, 2006). The mini planner wrapper can also return cost statistics for Web sources to the federated engine to aid in query optimization. These cost statistics, as described in DB2 Information Center, are:

1. The cardinality of a nickname. This is defined as the number of rows contained in the nickname (default 1000 rows).
2. The setup cost for a nickname. Setup cost represents the typical time, in milliseconds, that it takes a wrapper to get a query fragment ready to submit to the remote source (default 25 milliseconds).
3. The submission cost for a nickname. Submission cost represents the typical time, in milliseconds, that it takes a wrapper to submit a query fragment to the remote source (default 2000 milliseconds).
4. The advance cost for a nickname. This is the typical time, in milliseconds, that it takes to fetch a single row for the nickname (default 50 milliseconds).

Among these cost statistics, the set up and submission cost can be easily figured out, but the cardinality and the advance cost for a nickname are not easy to calculate for dynamic Web sources. We can, however, estimate the cardinality and advance cost for a nickname by keeping time statistics and cardinality information of previously executed CSQs on the same underlying relation. The estimation process can be initiated by starting with a conservative default time estimate and then improving on it using time statistics on recently executed CSQs on the same underlying relation.

RELATED WORK AND DISCUSSION

Our general strategy for querying Web sources in a data federation system fundamentally differs from other studies (see Florescu, Levy, & Mendelzon, 1998, for a review) in the same area for two reasons:

1. We clearly separate knowledge from code in wrapper development, and improve wrapper development speed and ease of maintenance.
2. We do not assume that we have the liberty to recode the existing federated database systems; thus we focus on improving the process of including and querying Web sources in cooperation with the existing data federation planners.

The majority of the studies in the area are concerned with query planning under source capability restrictions, and we find two types of approaches in the existing literature: 1) the black-box approach of pushing the capability handling to the wrapper level, and 2) the central planning approach by using a complex declarative language to describe capability restrictions. The IBM DB2 follows the first approach: handling capability restrictions is pushed down to the wrapper layer and it relies on Request-Reply-Compensate protocol to communicate with the wrappers. Although this is a generic framework to incorporate many different sources, coding a different wrapper every time for a Web site with different capability restrictions can be extremely wasteful and error-prone, since most of the code between these wrappers will be common.

There are projects that follow the second approach by describing capability restrictions with a declarative yet complex language. Examples of research projects, which more or less take this route, are Garlic Project at IBM (Roth

Table 2. Code and knowledge separation in Web wrapper development

	Extraction knowledge	Planning knowledge
Cooperative Planning Approach	Embedded in code	Embedded in code
Central Planning Approach	Embedded in code	Declarative
Our approach	Declarative	Declarative

& Schwarz, 1997; Papakonstantinou, Gupta, & Hass, 1998; Hass, Kossman, Wimmers, & Yang, 1997), TSIMMIS Project at Stanford (Chawathe, Garcia-Molina, Hammer, Ireland, Papakonstantinou, Ullman, & Widom, 1994), Information Manifold (Levy, Rajaraman, & Ordille, 1996), and DISCO (Tomasic, Raschid, & Valduriez, 1998). While this approach is more generic, it has not found its way into existing data federation technologies – perhaps due to its complexity.

The approach we take is a hybrid of these two. As in the black box approach, we push the capability handling to the wrapper level, and like the central planning approach we use declarative capability records. Yet these capability records are designed only to handle Web source access limitations and are not as general as the approaches found in the literature. This restriction simplifies the development of the query planner. Furthermore, our mini query planner creates query plans in cooperation with the central federated query planner, and thus differs from the central planning approach, which does not cooperate with the individual sources.

Another major difference we present is the clear separation of extraction and planning knowledge from the code. This is summarized in Table 2. The wrapper developer only deals with the task of specifying extraction and capability knowledge, and is not involved with low level coding as in other approaches.

Do not Web Services Solve the Problem?

It may be mistakenly thought that the solution offered here would not be needed if the Web sources were Web services returning XML. In fact, we are able to create virtual Web services from any semistructured Web source by using a version of the Cameleon# Web wrapping tool. The capability restrictions, however, are still valid problems for Web services, which often require input attributes before returning any results (Petropoulos, Deutsch, Papakonstantinou, & Katsis, 2007). There is an extra benefit of using Web services, as the capability restrictions could be automatically deduced from the Web service description language (WSDL) document instead of declaring them in the nickname statements. All the query dependency issues for arbitrary Web sources, however, equally apply to Web services as well. In fact, the built-in IBM wrapper for Web services prohibits the formulation of queries where dependencies create problems. Our solution is more general and can be used for Web services without artificial restrictions.

CONCLUSION

The Web is undoubtedly the largest and most diverse repository of data; unfortunately it was not designed to offer the capabilities of traditional database management systems. Modern databases promise to include Web sources in a data federation via "wrappers" so that they can be queried as if they are part of a single large database. There are still, however, significant hurdles to fulfilling this promise. With this study we introduced an improved way of dealing with Web source wrappers in federated database applications. With this new general strategy not only do we accelerate the inclusion of Web sources in federated databases,

but also we are able to eliminate unnecessary query restrictions. Our contribution is not only at a conceptual level, but also has been implemented using IBM's commercial database engine DB2. Most importantly, all of this has been achieved via extensions allowed by the federation engine, and without requiring any implementation changes in the existing data federation technology.

REFERENCES

Alatovic, T. (2001). *Capabilities aware, planner, optimizer, executioner for Context Interchange project*. Unpublished master's thesis, Massachusetts Institute of Technology, Cambridge, MA, USA.

Braumandl, R., Keidl, M., Kemper, A., Kossmann, D., Kreutz, A., Seltzsam, S., & Stocker, K. (2001). ObjectGlobe: Ubiquitous query processing on the Internet. *The VLDB Journal*, *10*(1), 48-71.

Chawathe, S.S., Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J.D., & Widom, J. (1994). The TSIMMIS project: Integration of heterogeneous information sources. In *Proceedings of the 16th Meeting of the Information Processing Society of Japan*, Tokyo, Japan (pp. 7–18).

Firat, A., Madnick, S., Yahaya, N., Kuan., C., & Bressan, S. (2005). *Information aggregation using the Caméléon# Web wrapper* (LNCS 3590, pp. 76-86).

Firat, A., Madnick, S., & Siegel, M. (2000). The Caméléon Web wrapper engine. In *Proceedings of the VLDB2000 Workshop on Technologies for E-Services* (pp. 1-9).

Florescu, D., Levy, A., & Mendelzon, A. (1998). Database techniques for the World Wide Web: A survey. *SIGMOD Record*, *27*(3), 59–74.

Fynn, K. (1997). *A planner/optimizer/executioner for context mediated queries*. Unpublished master's thesis, Massachusetts Institute of Technology, Cambridge, MA, USA.

Geer, D. (2003). Federated approach expands database-access technology. *Computer*, *36*(5), 18-20.

Haas, L.M., Kossmann, D., Wimmers, E.L., & Yang, J. (1997). Optimizing queries across diverse data sources. In M. Jarke, M.J. Carey, K.R. Dittrich, F.H. Lochovsky, P. Loucopoulos, & M.A. Jeusfeld (Eds.), *Proceedings of the 23rd International Conference on Very Large Data Bases* (pp. 276-285). San Francisco: Morgan Kaufmann Publishers,.

Haas, L.M., Lin, E.T., & Roth, M.A. (2002). Data integration through database federation. *IBM Systems Journal*, *41*(4), 578-596.

IBM. (2006, March 14). *DB2 information center*. Retrieved January, 31, 2007, from http://publib.boulder.ibm.com/infocenter/db2luw/v8//index.jsp

IBM. (2004, September 8). *DB2 information integrator wrapper developer's guide*. Retrieved January, 31, 2007, from http://publibfp.boulder.ibm.com/epubs/pdf/c1891740.pdf

Kache, H., Han, W., Markl, V., Raman, V., & Ewen, S. (2006). POP/FED: Progressive query optimization for federated queries in DB2. In U. Dayal, K. Whang, D. Lomet, G. Alonso, G. Lohman, M. Kersten, S.K. Cha, & Y. Kim (Eds.), *Proceedings of the 32nd International Conference on Very Large Data Bases* (pp. 1175-1178). San Francisco: Morgan Kaufmann Publishers

Kossmann, D. (2000). The state of the art in distributed query processing. *ACM Computing Surveys, 32*(4), 422-469.

Levy, A.Y., Rajaraman, A., & Ordille, J.J. (1996). Querying heterogeneous information sources using source descriptions. In T.M. Vijayaraman, A.P. Buchmann, C. Mohan, & N. L. Sarda (Eds.), *Proceedings of the 22th International Conference on Very Large Data Bases* (pp. 251-262). San Francisco: Morgan Kaufmann Publishers.

Li, C., & Chang, E. (2000). Query planning with limited source capabilities. In *Proceedings of the 16th International Conference on Data Engineering* (p. 401). Washington, DC: ICDE, IEEE Computer Society.

Li, C. (2003). Computing complete answers to queries in the presence of limited access patterns. *The VLDB Journal*, *12*(3), 211–227.

Papakonstantinou, Y., Gupta, A., & Haas, L. (1998). Capabilities-based query rewriting in mediator systems. *Distributed Parallel Databases, 6*(1), 73-110.

Pentaris, F., & Ioannidis, Y. (2006). Query optimization in distributed networks of autonomous database systems. *ACM Transactions on Database Systems, 31*(2), 537-583.

Petropoulos, M., Deutsch, A., Papakonstantinou, Y., & Katsis, Y. (2007). Exporting and interactively querying

Web service-accessed sources: The CLIDE system. *ACM Transactions on Database Systems, 32*(4), Article 22.

Roth, M.T., & Schwarz, P.M. (1997). Don't scrap it, wrap it! A wrapper architecture for legacy data sources. In M. Jarke, M.J. Carey, K.R. Dittrich, F.H. Lochovsky, P. Loucopoulos, & M.A. Jeusfeld (Eds.), *Proceedings of the 23rd International Conference on Very Large Data Bases* (pp. 266-275). San Francisco: Morgan Kaufmann Publishers.

Sheth, A.P., & Larson, J.A. (1990). Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, *22*(3), 183-236.

Suciu, D. (2002). Distributed query evaluation on semistructured data. *ACM Transactions on Database Systems, 27*(1), 1-62.

Somani, A., Choy, D., & Kleewein, J.C. (2002). Bringing together content and data management systems: Challenges and opportunities. *IBM System Journal*, *41*(4), 686-696.

Thiran, P., Hainaut, J., Houben, G., & Benslimane, D. (2006). Wrapper-based evolution of legacy information systems. *ACM Transactions on Software Engineering Methodologies, 15*(4), 329-359.

Tomasic, A., Raschid, L., & Valduriez, P. (1998). Scaling access to heterogeneous data sources with DISCO. *IEEE Transactions on Knowledge and Data Engineering*, *10*(5), 808–823..

Zadorozhny, V., Raschid, L., Vidal, M.E., Urhan, T., & Bright, L. (2002). Efficient evaluation of queries in a mediator for Web sources. In *Proceedings of the 2002 ACM SIGMOD international Conference on Management of Data* (pp. 85-96). New York: ACM.

Aykut Firat is an assistant professor in the information, operations, and analysis group at Northeastern University. Professor Firat received his PhD in management science from MIT Sloan School of Management, his MSc in systems analysis from Miami University, OH, and BSc in industrial engineering from Bogazici University, Istanbul. Professor Firat has primary research interest in understanding the technology, strategy, and organizational factors in integrating information systems. In particular, he is interested in heterogeneous database integration, and achieving semantic interoperability among Web sources and services in the emerging area of Semantic Web.

Stuart E. Madnick is the John Norris Maguire professor of Information Technology, Sloan School of Management and Professor of Engineering Systems, School of Engineering at the Massachusetts Institute of Technology. He has been a faculty member at MIT since 1972. He has served as the head of MIT's Information Technologies Group for more than twenty years. Dr. Madnick is the author or co-author of over 250 books, articles, or reports including the classic textbook, Operating Systems, and the book, The Dynamics of Software Development. His current research interests include connectivity among disparate distributed information systems, database technology, software project management, and the strategic use of information technology. He is presently co-director of the PROductivity From Information Technology Initiative and co-Heads the Total Data Quality Management research program. He has been active in industry, as a key designer and developer of projects such as IBM's VM/370 operating system and Lockheed's DIALOG information retrieval system. He has served as a consultant to corporations, such as IBM, AT&T, and Citicorp. He has also been the founder or co-founder of high-tech firms, including Intercomp, Mitrol, and Cambridge Institute for Information Systems, iAggregate.com and currently operates a hotel in the 14th century Langley Castle in England. Dr. Madnick has degrees in electrical engineering (BS and MS), management (MS), and computer science (PhD) from MIT. He has been a Visiting Professor at Harvard University, Nanyang Technological University (Singapore), University of Newcastle (England), Technion (Israel), and Victoria University (Australia).

Lynn Wu is a PhD candidate at MIT's Sloan School of Management. She is interested in studying the role of information and information technology in the productivity and performance of firms. Previously, she was a researcher at IBM. Lynn received a Bachelor's and a Master's degree from Electrical Engineering and Computer Science Department at MIT, along with a Bachelor's degree in finance from the MIT's Sloan School of Management.