



US007472137B2

(12) **United States Patent**
Edelstein et al.

(10) **Patent No.:** **US 7,472,137 B2**

(45) **Date of Patent:** **Dec. 30, 2008**

(54) **DATA QUERY AND LOCATION THROUGH A CENTRAL ONTOLOGY MODEL**

(58) **Field of Classification Search** 707/1-10,
707/100-104.1, 200-206
See application file for complete search history.

(75) Inventors: **Joseph Edelstein**, Jerusalem (IL);
Hayden Marchant, Ramat Beit
Shemesh (IL); **Rannen Meir**, Jerusalem
(IL); **Marcel Zvi Schreiber**, Jerusalem
(IL)

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,499,371	A	3/1996	Henninger et al.
6,374,252	B1	4/2002	Althoff et al.
6,591,272	B1	7/2003	Williams
6,643,633	B2	11/2003	Chau et al.
6,687,873	B1 *	2/2004	Ballantyne et al. 715/500
6,732,109	B2	5/2004	Lindberg et al.
6,847,974	B2	1/2005	Wachtel

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 363 days.

(Continued)

(21) Appl. No.: **11/182,455**

OTHER PUBLICATIONS

Irani, Romin, "Enabling Web Services BEA WebLogic" Sep. 26, 2001, pp. 1-4. <http://www.webservicesarchitect.com/content/articles/irrani05.asp>.

(22) Filed: **Jul. 15, 2005**

(Continued)

(65) **Prior Publication Data**

US 2006/0167927 A1 Jul. 27, 2006

Primary Examiner—Isaac M Woo

(74) *Attorney, Agent, or Firm*—Duke W. Yee; Doug H. Lefevre; Theodore D. Fay, III

Related U.S. Application Data

(60) Division of application No. 10/159,516, filed on May 31, 2002, now abandoned, which is a continuation-in-part of application No. 10/104,785, filed on Mar. 22, 2002, now Pat. No. 7,146,399, which is a continuation-in-part of application No. 10/053,045, filed on Jan. 15, 2002, now abandoned, which is a continuation-in-part of application No. 09/904,457, filed on Jul. 6, 2001, now Pat. No. 7,093,200, which is a continuation-in-part of application No. 09/866,101, filed on May 25, 2001, now Pat. No. 7,099,885.

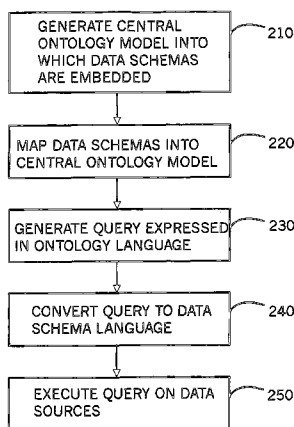
(57) **ABSTRACT**

A data query system including a first storage medium including a data schema having a data schema query language associated therewith, a second storage medium including an ontology model including classes and properties, the ontology model having an ontology query language associated therewith, wherein constructs of the database schema are mapped to corresponding classes, properties or compositions of properties of the ontology model, and an ontology query processor generating a query expressed in the data schema query language corresponding to a specified query expressed in the ontology query language. A method is also described and claimed.

(51) **Int. Cl.**
G06F 17/00 (2006.01)

(52) **U.S. Cl.** **707/104.1; 707/101; 707/102; 707/103 R**

4 Claims, 14 Drawing Sheets



U.S. PATENT DOCUMENTS

6,947,943 B2 9/2005 DeAnna et al.
6,985,905 B2 1/2006 Prompt et al.
7,007,029 B1 2/2006 Chen
2003/0050932 A1 3/2003 Pace et al.

OTHER PUBLICATIONS

Klein et al., "The Relation between Ontologies and schema languages" Vrije Universiteit Amsterdam, <http://www.cs.vu.nl/~mcklein/papers/oil-xmls.pdf>.

* cited by examiner

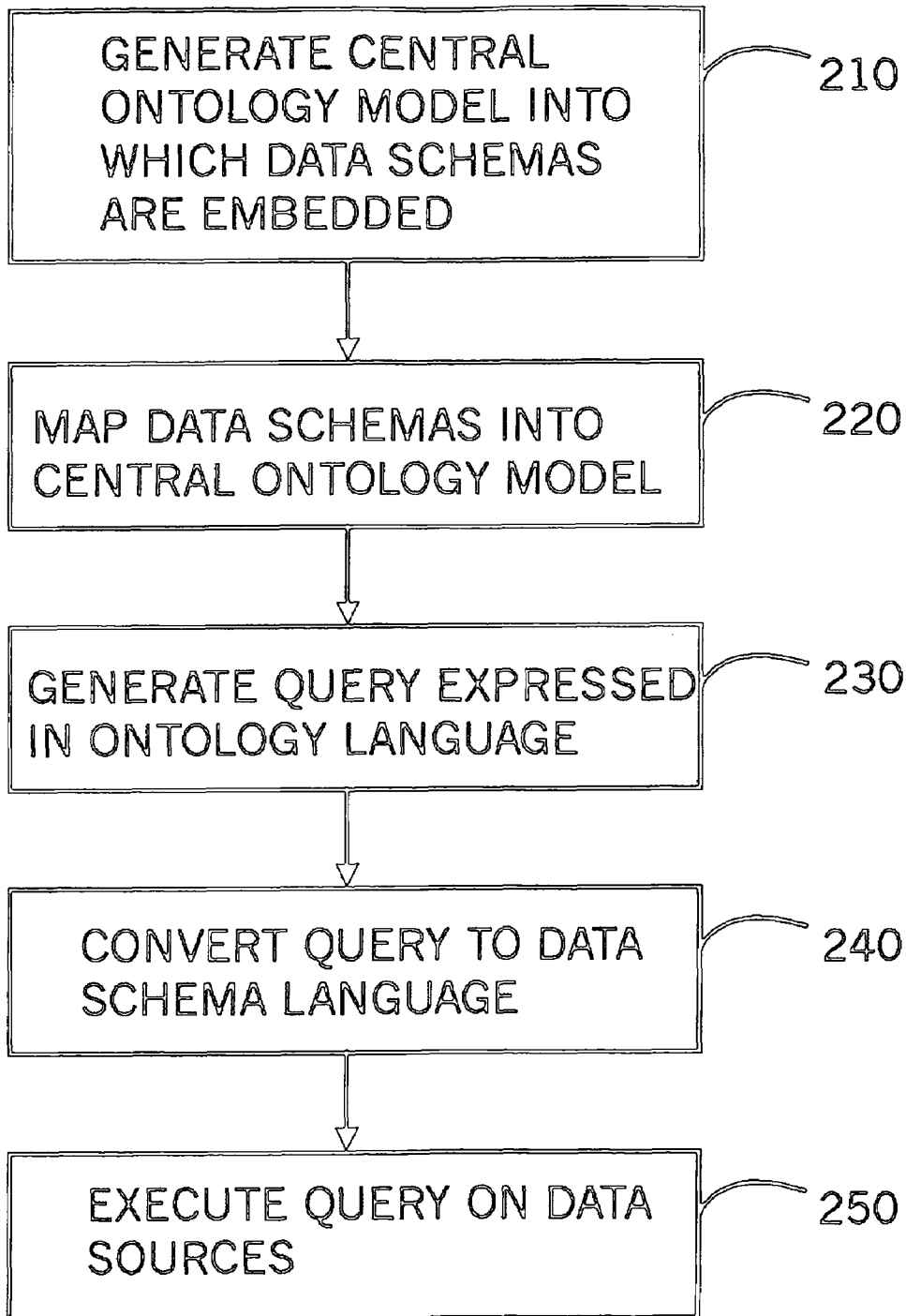


FIG. 2

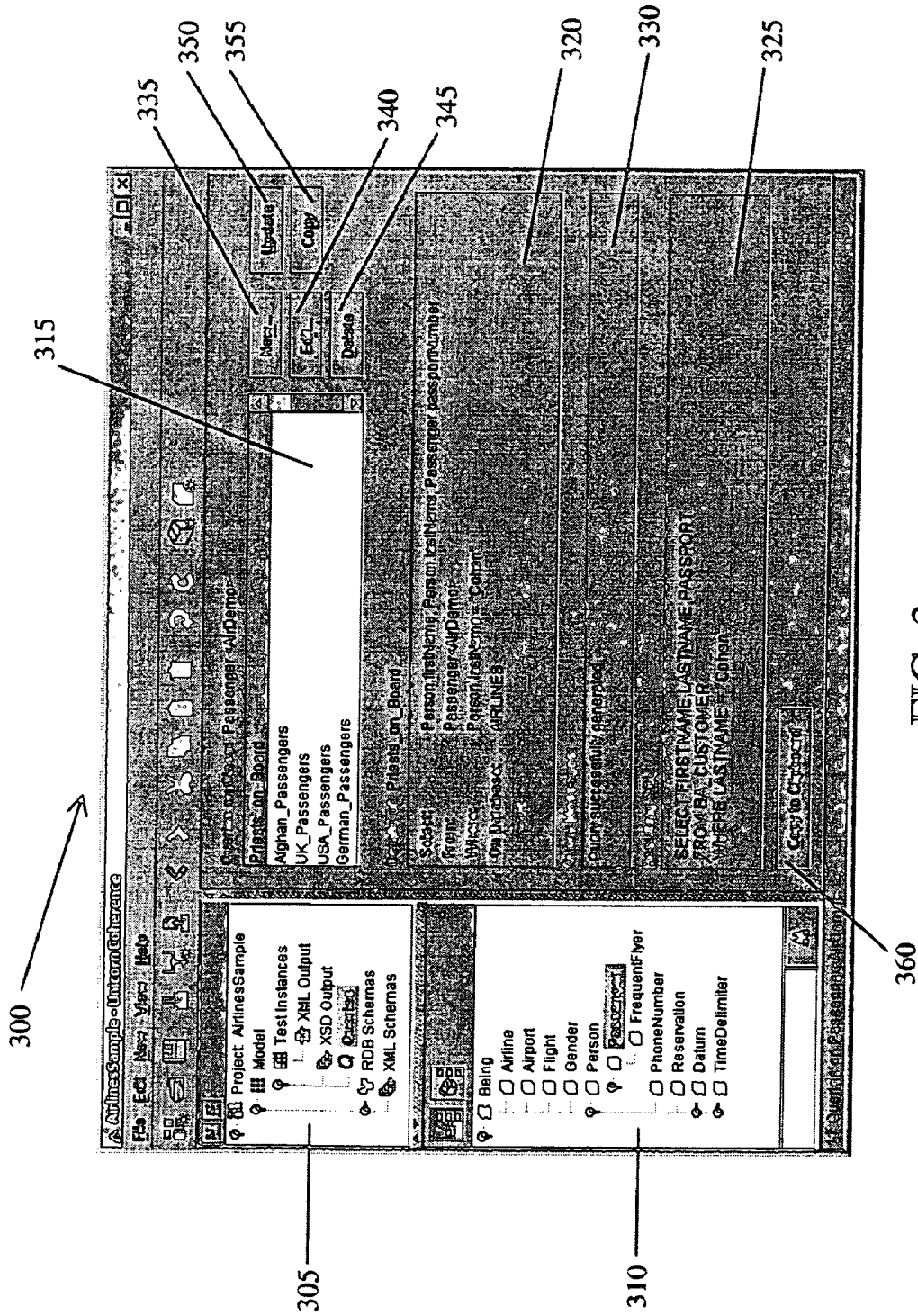


FIG. 3

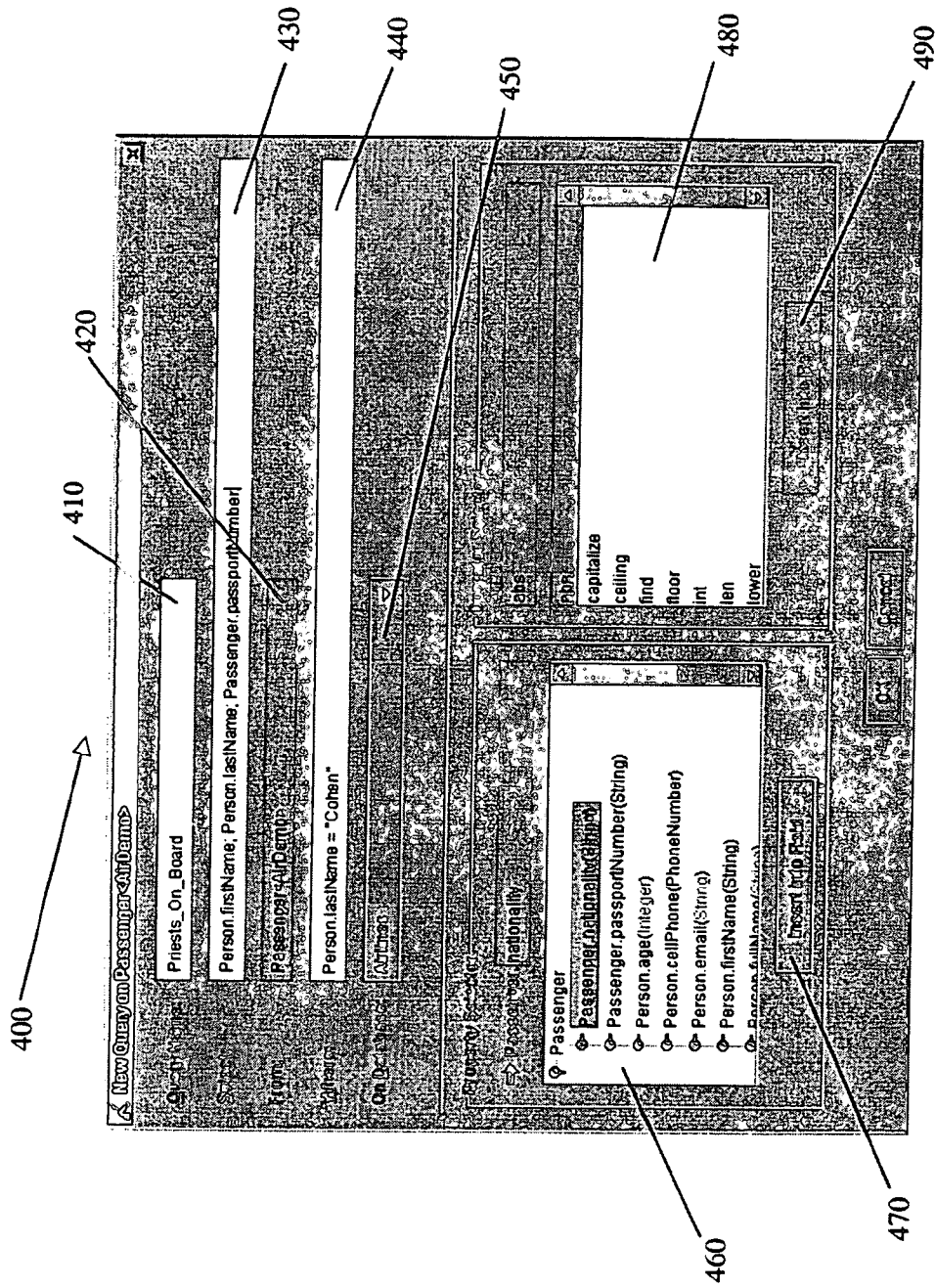


FIG. 4

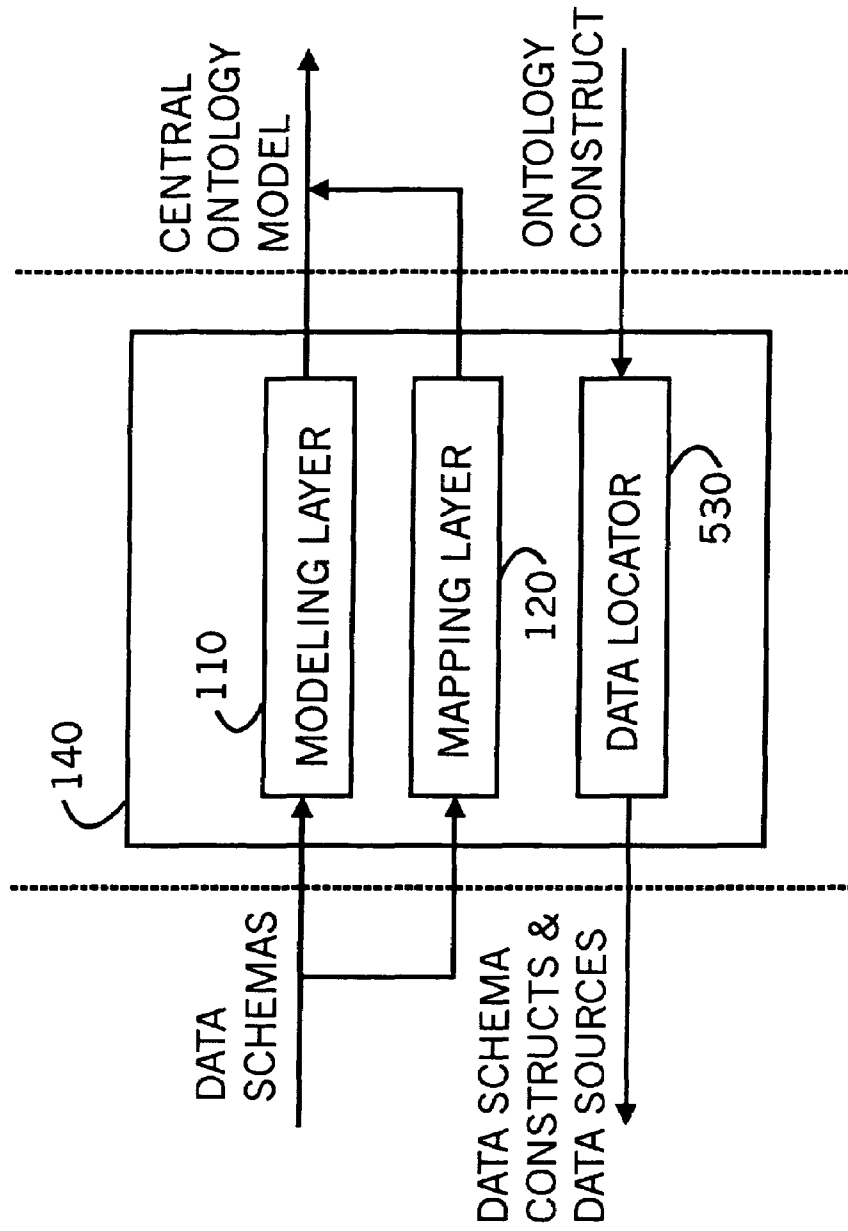


FIG. 5

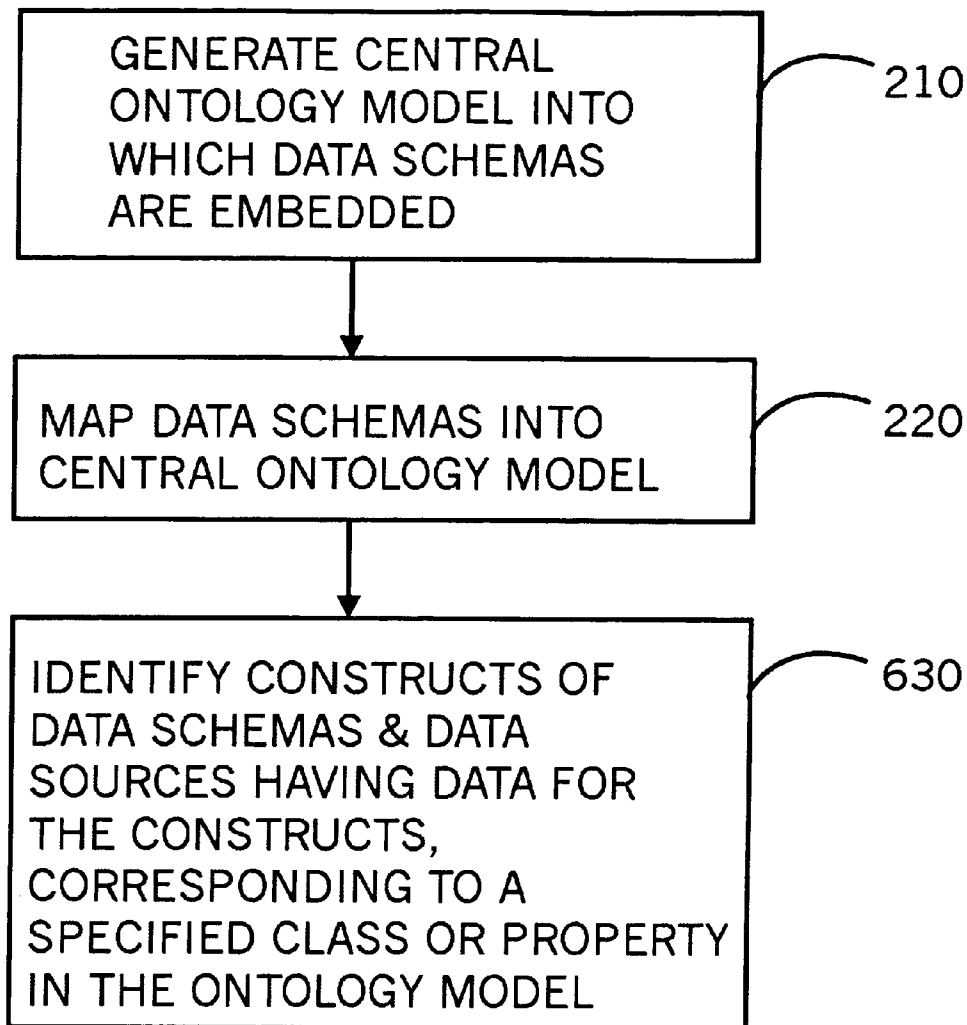


FIG. 6

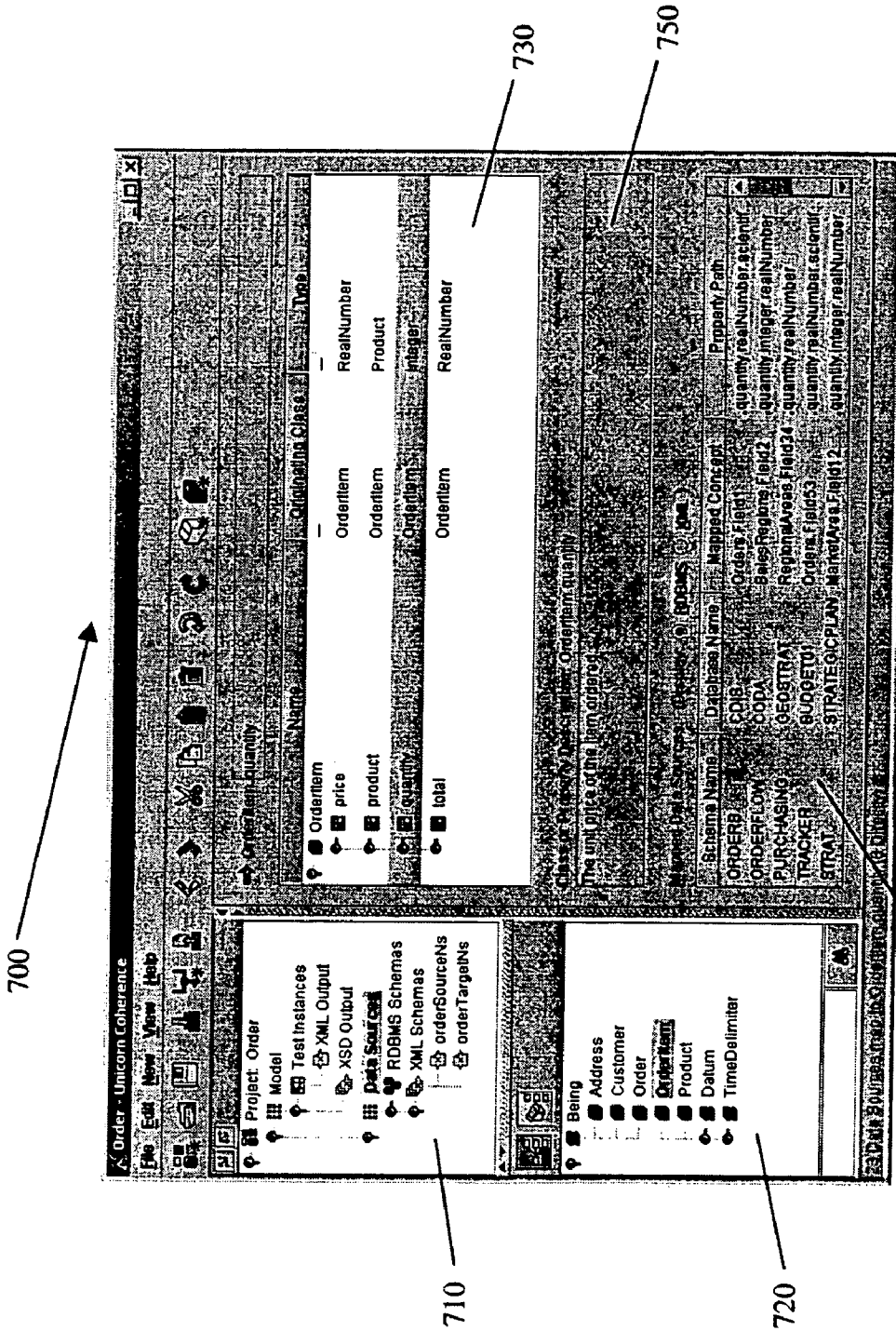


FIG. 7

Mappings To Class: Order

Package: <Orders>

RDBMS Schemas

Order Mapped to: 2 Sources
Schema Name: Database Name
Shipping: Logistics
Mapped Table: PurchaseOrder

Order.customer Mapped to: 5 Sources (2 directly)
Schema Name: Database Name
Shipping: Logistics
Mapped Concept: FK:FK_PurchaseOrder_PurchaseOrderEntry
Property Path: Order.customer (Customer)

Order.date Mapped to: 0 Sources
Not Mapped

Order.id Mapped to: 3 Sources (3 directly)
Schema Name: Database Name
Shipping: Logistics
Mapped Concept: PurchaseOrder.id
Property Path: Order.id (Integer)

Order.items Mapped to: 3 Sources (3 directly)
Not Mapped

Order.total Mapped to: 3 Sources (2 directly)
Schema Name: Database Name
Shipping: Logistics
Mapped Concept: PurchaseOrder.totalcost
Property Path: Order.total.scientificRepresentation (String)

XML Schemas

Order Mapped to: 2 Sources
Schema Name: Concept
orderTargetNS: Complex Type
Concept X-Path: /Shipment

FIG. 8

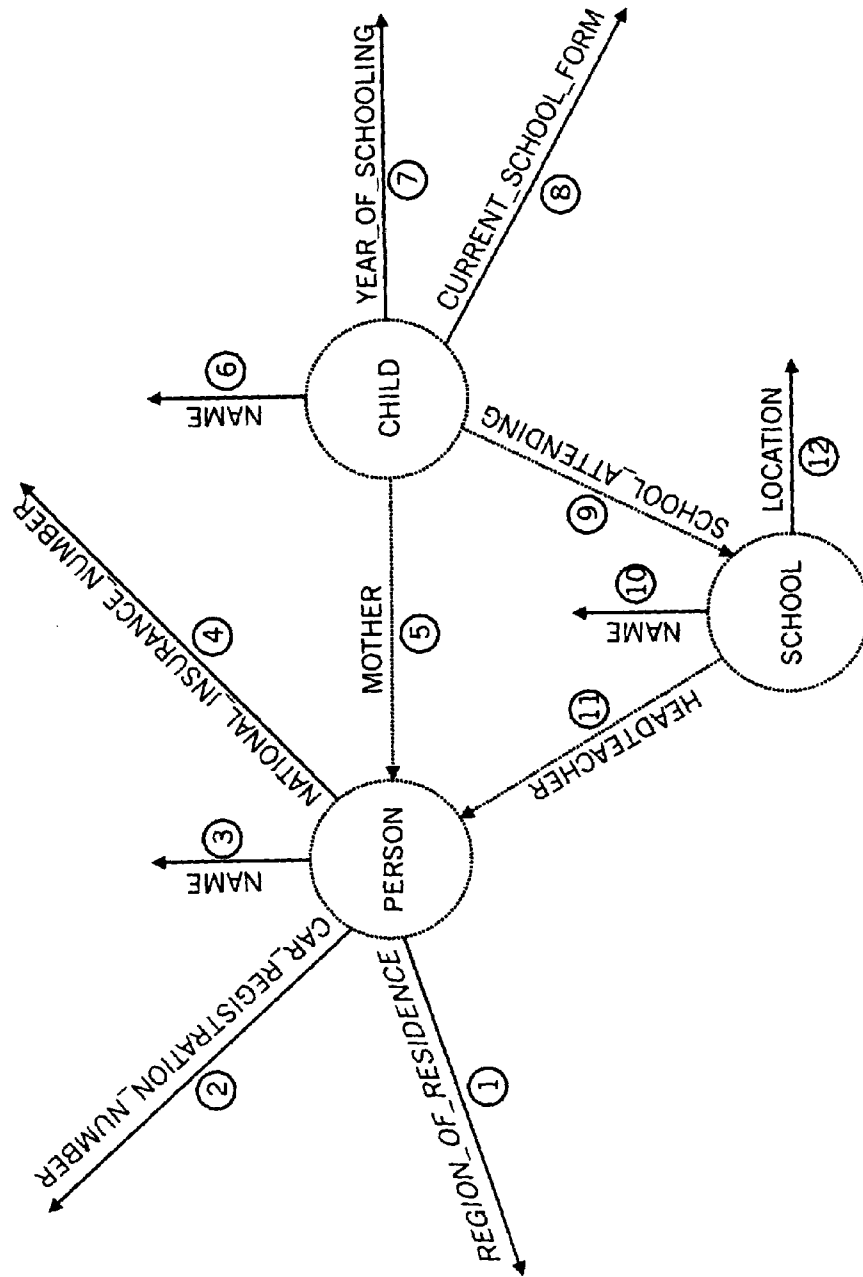


FIG. 9

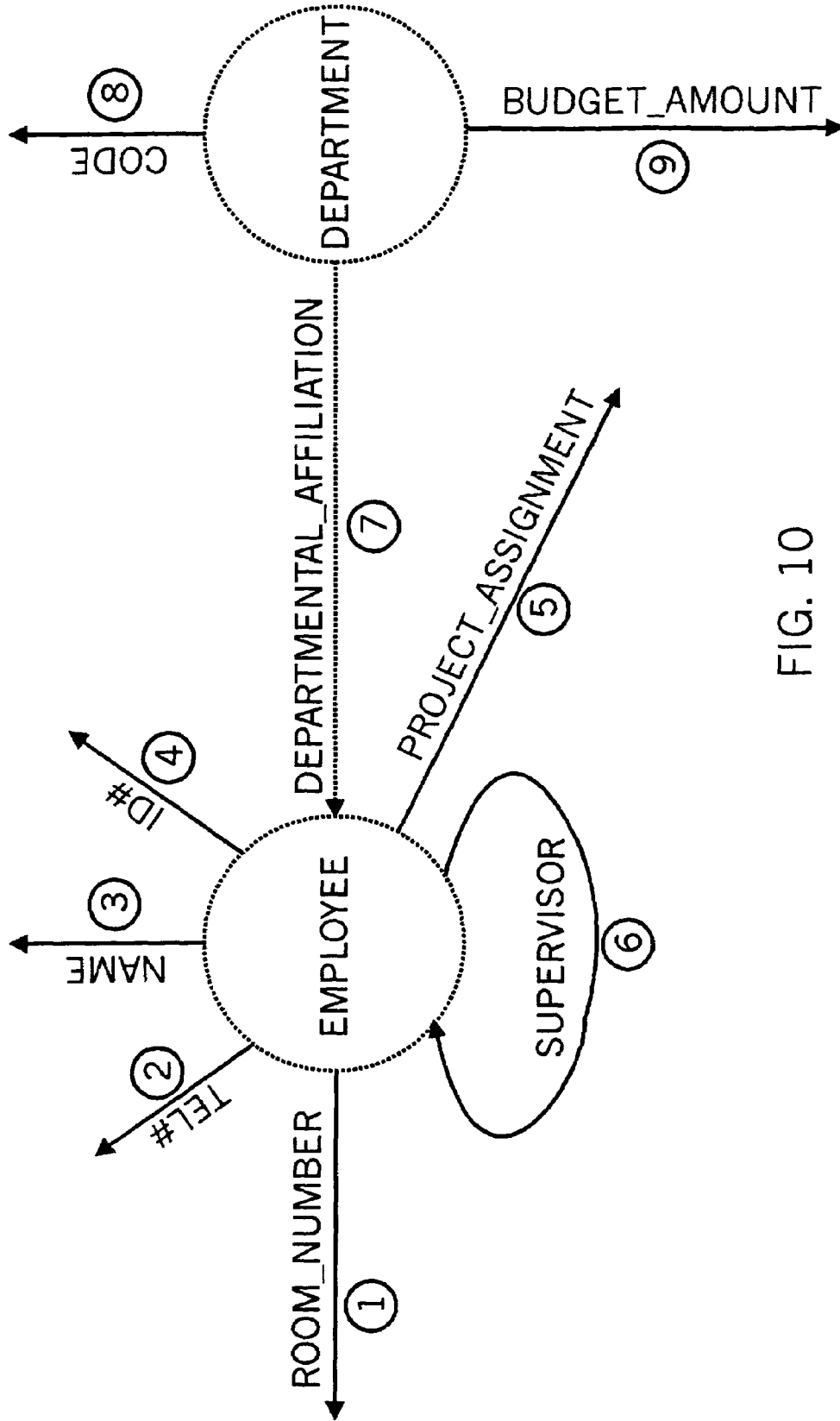


FIG. 10

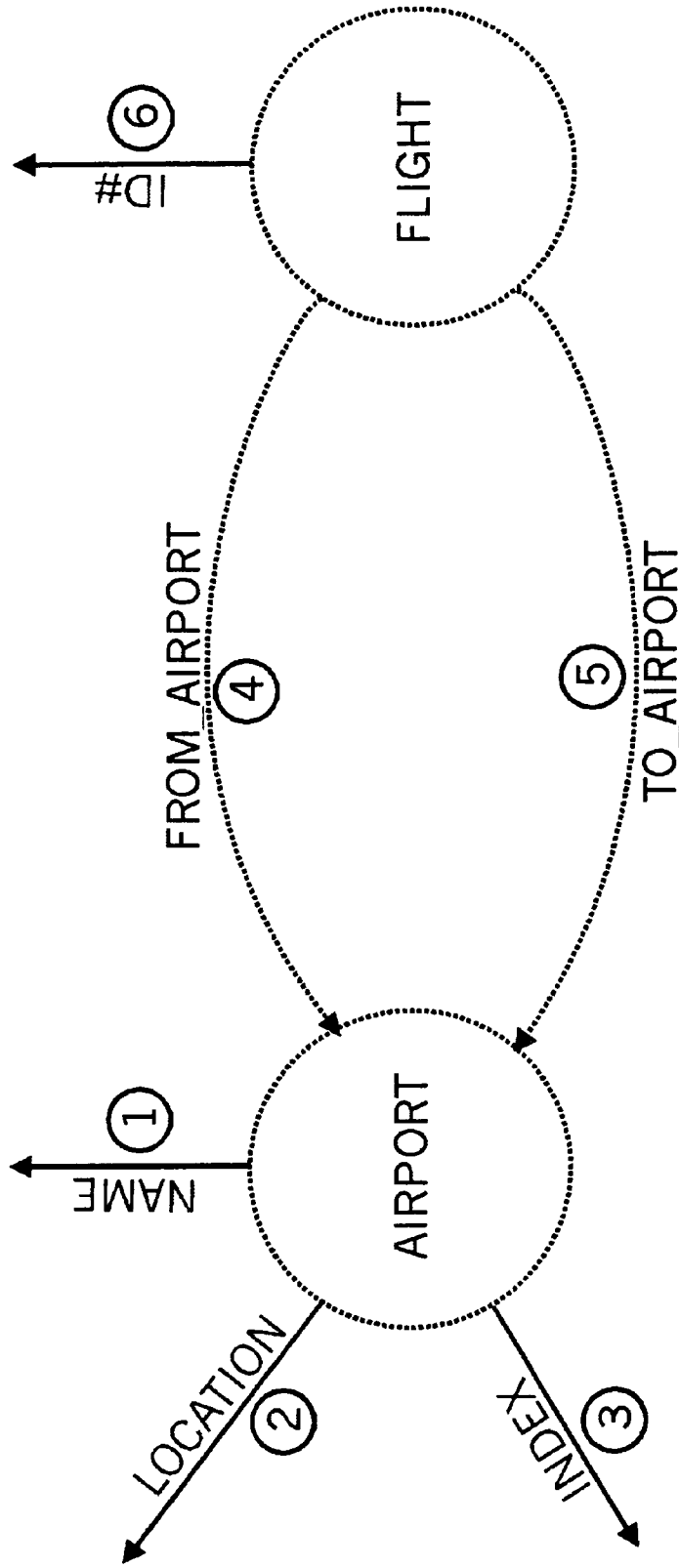


FIG. 11

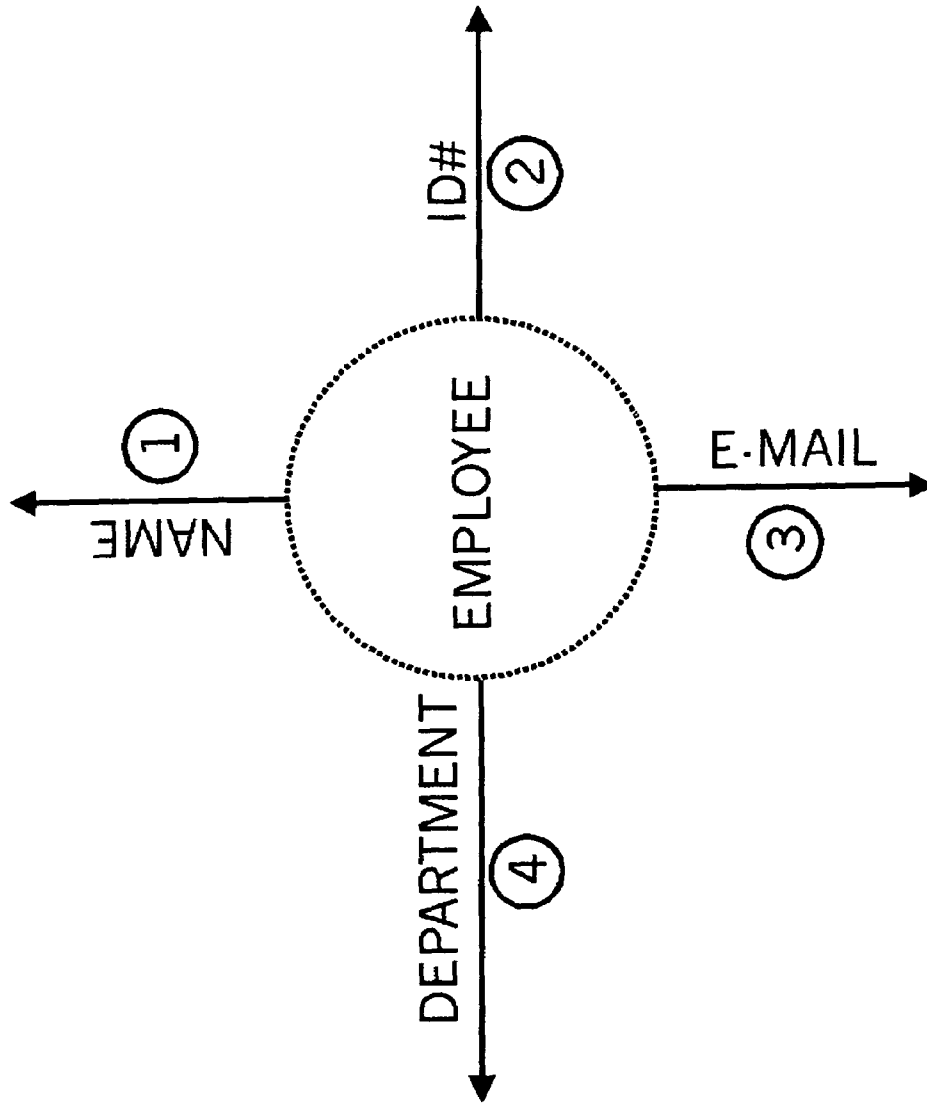


FIG. 12

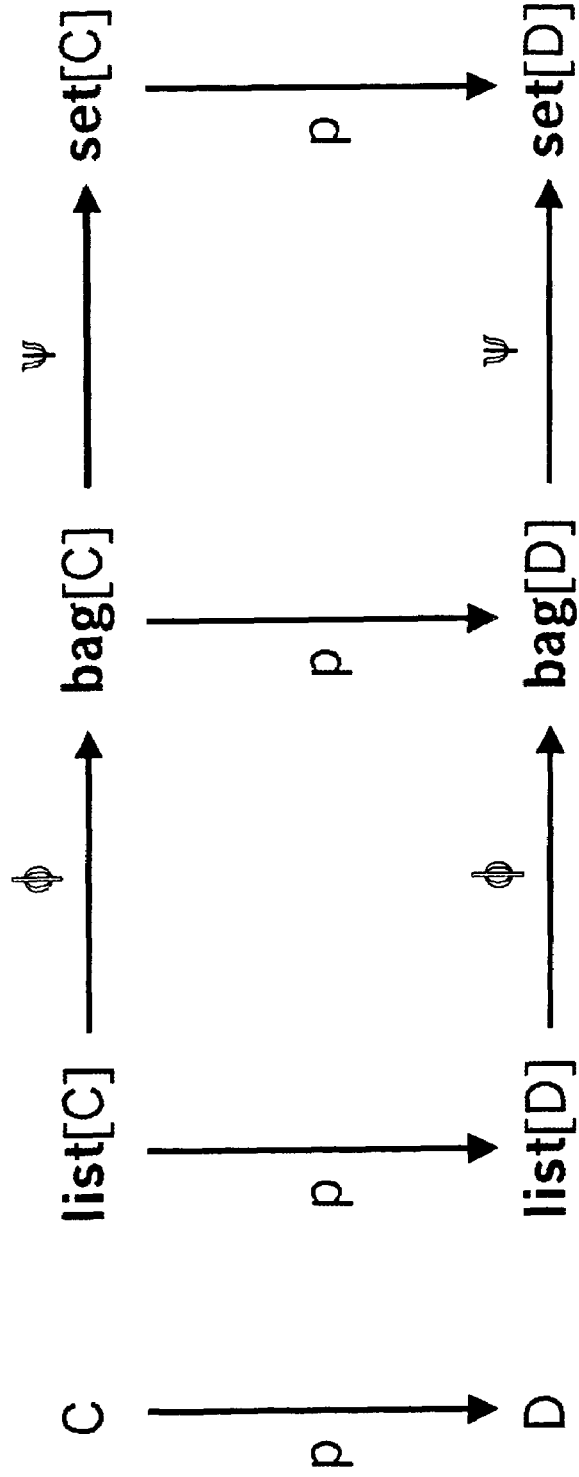


FIG. 13

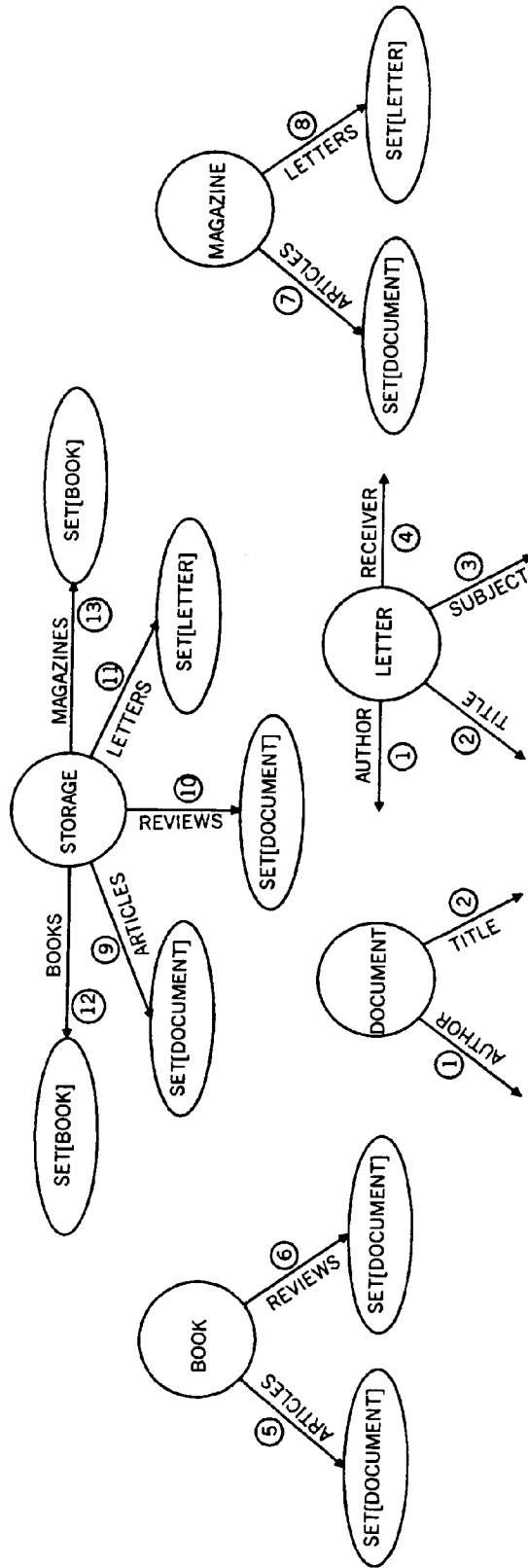


FIG. 14

DATA QUERY AND LOCATION THROUGH A CENTRAL ONTOLOGY MODEL

CROSS REFERENCES TO RELATED APPLICATIONS

This application is a divisional application of U.S. patent application Ser. No. 10/159,516, filed on May 31, 2002 now abandoned which is a continuation-in-part of assignee's application U.S. Ser. No. 10/104,785, filed on Mar. 22, 2002 now U.S. Pat. No. 7,146,399, entitled "Run-Time Architecture for Enterprise Integration with Transformation Generation," which is a continuation-in-part of application U.S. Ser. No. 10/053,045, filed on Jan. 15, 2002 now abandoned, entitled "Method and System for Deriving a Transformation by Referring Schema to a Central Model," which is a continuation-in-part of assignee's application U.S. Ser. No. 09/904,457 filed on Jul. 6, 2001 now U.S. Pat. No. 7,093,200, entitled "Instance Browser for Ontology," which is a continuation-in-part of assignee's application U.S. Ser. No. 09/866,101 filed on May 25, 2001 now U.S. Pat. No. 7,099,885, entitled "Method and System for Collaborative Ontology Modeling."

FIELD OF THE INVENTION

The present invention relates to data processing; specifically to querying of data sources and locating data within data sources.

BACKGROUND OF THE INVENTION

Ontology provides a way to model things that exist. Basic constructs of an ontology model are classes, properties thereof and inheritance. Classes are sets, the elements of which are referred to as instances of the class. For example, a class *People* is a set of instances that represent specific people. A property, *p*, of a class is a function $p: C \rightarrow D$ from a class *C*, referred to as the source of *p*, to a class *D*, referred to as the target of *p*. The classes *C* and *D* may be the same class of different classes. When it is important to distinguish between properties defined on different classes, the notation *C.p* is used to denote a property, *p*, defined on *C*.

Properties may be composed, so that if $p: C \rightarrow D$ and $q: D \rightarrow E$, then the composition $q \circ p: C \rightarrow E$ has source *C* and target *E*. The composition is denoted by *C.p.q*.

A class *C* is said to be a subclass of *D* if $C \subset D$, in which case *D* is also said to be a superclass of *C*. In this case, every instance of *C* is also an instance of *D*, and properties defined on *D* are also defined on *C* by inheritance. For example, a class named *Passengers* may be a subclass of a class named *People*. A property such as *firstName*, defined on *People* is inherited by *Passengers*.

A special class named *Being* is defined in an ontology model as a universal class that contains all classes as subclasses thereof. Properties defined on *Being* are thus inherited by all classes in the ontology model.

Certain properties, referred to as "representations," take on concrete fundamental alphanumeric values. The significance of representations is that they are properties one can reason about using arithmetic, logical and string operators, since their type corresponds to the types of mathematical expressions and programming language expressions.

In order to accommodate and provide values for representations, a special class *Values* is preferably created, so as to include all possible fundamental values a property may have. In addition, a special class *Formats* is also created, to include

formats in which instances of *Values* can be expressed. Formats include inter alia conventional integer formats, real number formats, character string formats and date and time formats. A function representation: $Values \times Formats \rightarrow Alphanumerics$, converts a value into an alphanumeric string according to a specific format. For example, if $lastName: People \rightarrow Values$, then $representation(person.lastName, titleCase) = "Smith"$ (a character string), for an instance, *person*, of *People* corresponding to John Smith. Observe that *lastName* is a representation, and *titleCase* is a format.

Alternatively, various formats can be modeled as properties on the class *Values*, or subclasses thereof. With respect to this alternative model design choice, the last name of John Smith represented as a character string in title case is denoted *person.lastName.titleCase*. Observe that *lastName* and *titleCase* are both representations in this alternative model.

Applicant's co-pending application U.S. Ser. No. 10/053,045, filed on Jan. 15, 2002 and entitled "Method and System for Deriving a Transformation by Referring Schema to a Central Model" describes mapping data schema, including inter alia relational database schema and XML schema, into a central ontology model. Basic constructs of the data schema are mapped to classes, properties and compositions of properties in the central ontology model. Thus, for relational database schema, tables are generally mapped to ontology classes, and fields of tables are generally mapped to ontology properties or compositions of properties—more specifically, to properties or compositions of properties with target *Values*. Similarly, for XML schema, complex types are generally mapped to ontology classes, and elements and attributes within complex types are generally mapped to ontology properties or compositions of properties.

Enterprise data systems, especially for large enterprises, typically include multiple data sources that may be compliant with different data schemas. Indeed, as a result of several generations of IT and/or mergers and acquisitions, several databases with different schemas may contain information on the same functional area of the business. For example, enterprise employment data may be stored in relational databases conforming to a first relational database schema, enterprise accounting data may be stored in relational databases conforming to a second relational database schema, enterprise sales forecasts may be stored in relational databases conforming to a third relational database schema, enterprise inventory data may be stored in XML documents conforming to a first XML schema, and enterprise bill of materials data may be stored in XML documents conforming to a second XML schema. Often these various data sources may overlap, and it is difficult for a user to query across the enterprise data sources. For example, a user may want to know the bills of materials for items that need to be replenished in inventory based on demand forecasts.

There is thus a need for a unified querying tool that enables a user to query across data sources conforming to disparate data schemas.

The need for a unified querying tool also arises with one or multiple data sources, when engineers involved in application development, enterprise application integration or data warehousing may not be aware of the precise semantics of a database, and may therefore be unable to use its data appropriately.

The need for a unified querying tool also arises when a single question crosses multiple data sources.

Another difficulty faced by enterprises is being able to locate data within multiple data sources. With reference to the example above, a user may want to locate data sources con-

taining employee stock option data. Such data may be distributed over multiple data sources, and may involve joining relational database tables that conform to different data schema.

There is thus a need for a data locator tool that enables a user to specify data of interest, and receive a list of constructs corresponding to the data of interest, and the various data sources containing data for such constructs.

The need for a data locator tool also arises when trying to locate overlaps, where a single aspect of enterprise information is stored in multiple locations. Such overlaps signal the potential for bad data quality, as they generally lead to inconsistencies.

SUMMARY OF THE INVENTION

Certain embodiments of the present invention provide a method and system for unified querying of data sources conforming to possibly disparate data schemas by use of a central ontology model into which the data schemas are embedded. In a preferred embodiment, the present invention uses an ontology query language for expressing queries about an ontology. Certain embodiments of the present invention enable a user to ask a question using an agreed business vocabulary for an ontology, and ensures that this question is translated accurately into a query on the underlying data sources.

Using mappings of the data schemas into the central ontology model, certain embodiments of the present invention convert a query referring to the central ontology model into a corresponding query referring to a data schema that is embedded within the central ontology model. Thus a user can build a generic query referring to the central ontology model, and convert it to queries that can be executed on various data sources. In a preferred embodiment, the present invention uses a query builder tool for generating queries referring to an ontology.

The present invention also provides a method and system for data location. In a preferred embodiment, a user can select a class or property from a central ontology model, and locate constructs in various data schema that correspond to his selection, as well as locations of data sources that contain data for such constructs.

Certain embodiments of the present invention enable a user to locate overlaps, where a single aspect of enterprise information is stored in multiple locations. The present invention can be used to identify ontology constructs that are mapped to too many different data sources. For example, the present invention can be used inter alia to highlight constructs that are mapped more than a given number, *n*, times.

There is thus provided in accordance with an embodiment of the present invention a data query system including a first storage medium including a data schema having a data schema query language associated therewith, a second storage medium including an ontology model including classes and properties, the ontology model having an ontology query language associated therewith, wherein constructs of the data schema are mapped to corresponding classes, properties or compositions of properties of the ontology model, and an ontology query processor generating a query expressed in the data schema query language corresponding to a specified query expressed in the ontology query language.

There is provided in accordance with an embodiment of the present invention a COBOL mapping system for embedding a COBOL Copy Book including group items and elementary items into an ontology model including classes and properties, including at least one data construct for mapping at least

one COBOL group item within a COBOL Copy Book to at least one class of an ontology model, and at least one data construct for mapping at least one COBOL elementary items within the COBOL Copy Book to at least one property or composition of properties of the ontology model.

There is further provided in accordance with an embodiment of the present invention a method for embedding a COBOL Copy Book including group items and elementary items into an ontology model including classes and properties, including mapping at least one COBOL group item within a COBOL Copy Book to at least one class of an ontology model, and at least one data construct for mapping at least one COBOL elementary items within the COBOL Copy Book to at least one property or composition of properties of the ontology model.

Other embodiments of the invention are also described, including, for example, computer readable medium (or media) containing executable computer program instructions, which when executed by a digital processing system, such as a general purpose computer having at least one microprocessor and memory and a bus, cause the system to perform one or more methods described herein. The medium may be a magnetic medium (e.g. a hard disk in a drive) or an optical medium (e.g. a CD-ROM).

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be more fully understood and appreciated from the following detailed description, taken in conjunction with the drawings in which:

FIG. 1 is a simplified block diagram of a data query system, in accordance with a preferred embodiment of the present invention;

FIG. 2 is a simplified flowchart of a method for querying a data source, in accordance with a preferred embodiment of the present invention;

FIG. 3 is an illustration of a user interface for converting a query expressed in an ontology query language to a corresponding query expressed in a data schema query language, in accordance with a preferred embodiment of the present invention;

FIG. 4 is an illustration of a user interface for generating a query expressed in an ontology query language, in accordance with a preferred embodiment of the present invention;

FIG. 5 is a simplified block diagram of a data locator system, in accordance with a preferred embodiment of the present invention;

FIG. 6 is a simplified flowchart of a method for locating data, in accordance with a preferred embodiment of the present invention;

FIG. 7 is an illustration of a user interface for locating data within at least one data source, in accordance with a preferred embodiment of the present invention;

FIG. 8 is an illustration of a visual display for displaying data location results, in accordance with a preferred embodiment of the present invention;

FIG. 9 is an illustration of ontology model corresponding to a first example;

FIG. 10 is an illustration of ontology model corresponding to a second example;

FIG. 11 is an illustration of ontology model corresponding to a third example;

FIG. 12 is an illustration of ontology model corresponding to a fourth example;

5

FIG. 13 is a commutative diagram illustrating consistency of multi-valued properties; and

FIG. 14 is an illustration of ontology model with multi-valued properties.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

Certain embodiments of the present invention concern querying of data sources, and data location within the sources, through use of a central ontology model. Specifically, certain embodiments of the present invention use an ontology query language for expressing queries relating to an ontology, and a query conversion tool for converting such queries into corresponding queries relating to data schemas that are embedded within the ontology. The present invention, in one embodiment, can enable a user to ask a question using an agreed business vocabulary for an ontology, and ensures that this question is translated accurately into a query on the underlying data sources.

The present invention also uses, in certain embodiments, a data locator for locating data corresponding to specified classes and properties of an ontology, within data sources that conform to data schemas that are embedded within the ontology.

Reference is now made to FIG. 1, which is a simplified block diagram of a data query system, in accordance with a preferred embodiment of the present invention. Shown in FIG. 1 is a modeling layer 110, which enables a user to generate a central ontology model into which one or more data schemas can be embedded. While generating the central ontology model, the user preferably also maps constructs of the data schemas into corresponding classes, properties and compositions of properties in the central ontology model, through use of a mapping layer 120. Modeling and mapping layers are described in applicant's co-pending application U.S. Ser. No. 10/053,045, referenced hereinabove, the contents of which are hereby incorporated by reference.

FIG. 1 is divided by dashed vertical lines into a left part, a middle part and a right part. The left part relates to the data schemas and data sources, and the right part relates to the central ontology. The middle part relates to an engine that maps items from the left part to the right part, or vice versa.

A query later 130 processes a query expressed in an ontology language and generates a corresponding query expressed in a data schema language. Together, modules 110, 120 and 130 may reside within a single computer 140 for local computing, or may be distributed among a plurality of computers for distributed computing.

A query processor 140 is used to execute the query expressed in the data schema language on one or more data sources. In a preferred embodiment of the present invention, the query expressed in a data schema language is generated through an ontology query builder 150.

Reference is now made to FIG. 2, which is a simplified flowchart of a method for querying .data source, in accordance with a preferred embodiment of the present invention. At step 210 a central ontology model into which one or more given data schemas are embedded is generated. At step 220 constructs of the given data schemas are mapped into classes, properties and compositions of properties of the central ontology model. At step 230 a query expressed in an ontology query language is generated. At step 240 the query expressed in the ontology query language is converted to a query expressed in a data schema language. At step 250 the query expressed in the data schema language is executed on one or more data sources.

6

Reference is now made to FIG. 3, which is an illustration of a user interface for converting a query expressed in an ontology query language to a corresponding query expressed in a data schema query language, in accordance with a preferred embodiment of the present invention. Shown in FIG. 3 is a window 300 containing various panes. An upper left pane 305 displays components of a project named AirlinesSample. Shown highlighted in pane 305 is a "Queries" component. A lower left pane 310 displays an ontology model in a class hierarchical view. Pane 310 indicates that all classes inherit from a superclass named Being, and that a class named Person contains a subclass named Passenger, which contains a further subclass named FrequentFlyer. Shown highlighted in pane 310 is the class Passenger, which is the subject class for queries are displayed in the right-hand panes.

Specifically, upper right pane 315 displays a list of various queries on the class Passenger that already exist. Shown highlighted in pane 315 is a query named Priests_on_Board. Each query listed in pane 315 is expressed in an ontology query language. A pane 320 displays the query highlighted in pane 315 in the ontology query language. In a preferred embodiment of the present invention, a query expressed in the ontology query language includes three clauses. A first clause, designated as SELECT indicates one or more properties to be displayed. A second clause, designated as FROM indicates one or more subject class on which the properties are defined. A third clause, designated as WHERE indicates one or more conditions satisfied by one or more values of properties defined on the subject class. The three clauses for the Priests_on_Board query are displayed in a pane 320. Those skilled in the art will appreciate that the SELECT, FROM and WHERE clauses resemble their SQL counterparts. In a preferred embodiment of the present invention, an ontology query also includes a clause indicating a database for which the query is intended, as shown in pane 320.

A bottom right pane 325 displays a query expressed in a data schema language, which corresponds to the query displayed in pane 320. Specifically, in FIG. 3 pane 325 displays an SQL query for the database AIRLINES, which corresponds to the query Priests_on_Board in pane 320. The system of the present invention preferably converts the query in pane 320 to the query in pane 325 using the mapping from the relational database schema AIRLINES into the ontology model AirlinesSample.

A pane 330 is used to display system messages regarding the process of conversion from the query of pane 320 to the query of pane 325.

Various buttons are provided in window 300 for user interaction. Buttons 335, 340, 345, 350 and 355 relate to pane 315 and the list of queries displayed therein. Button 335 enables a user to build a new query for the class Passenger. Upon clicking on button 335, a user interface for building queries is preferably invoked. Such a user interface is described with respect to FIG. 4 hereinbelow.

Button 340 enables a user to edit an existing query on the class Passenger. Button 345 enables a user to delete an existing query. Button 350 enables a user to update the SQL query displayed in pane 325 by re-executing the process that generates the SQL query from the ontology query. A button 355 enables a user to copy an existing query, for use inter alia in building a new query or editing an existing query. Finally, a button 360 is provided under pane 325 for copying a query displayed therein to a clipboard, preferably for input to a database application that executes the query on one or more data sources.

7

As can be seen in FIG. 3, the ontology language query
 SELECT: Person firstName, Person.lastName, Passenger.
 passportNumber
 FROM: Passenger<AirDemo>
 WHERE: Person.lastName="Cohen"
 ON DATABASE: Airlines

is converted to the SQL query
 SELECT FIRSTNAME, LASTNAME, PASSPORT
 FROM BA_CUSTOMER
 WHERE LASTNAME="Cohen"

on the database AIRLINES. The table BA_CUSTOMER corresponds to the ontology class Passenger, and the fields FIRSTNAME, LASTNAME and PASSPORT of table BA_CUSTOMER correspond respectively to the properties firstName, lastName and passportNumber of class Passenger.

Reference is now made to FIG. 4, which is an illustration of a user interface for generating a query expressed in an ontology query language, in accordance with a preferred embodiment of the present invention. Shown in FIG. 4 is a window 400 used for filling in the SELECT, FROM and WHERE clauses of a query on an ontology class. A field 410 is provided to specify a name for the query. Fields 420, 430 and 440 are provided to enter three clauses of the query. The entries in these fields correspond to the query displayed in pane 320 of FIG. 3. Field 450 is provided to specify one or more databases on which the query is intended to be applied. In a preferred embodiment, the system of the present invention converts the query specified in fields 420, 430, 440 to a database query on the one or more databases specified in field 450. Such a converted query is displayed in pane 325 of FIG. 3.

To assist a user in filling in fields 420, 430 and 440, a lower left pane 460 displays a list of properties defined on the class Passenger. It is noted that some of these properties, such as Person.firstName, are inherited from the superclass Person. Preferably, a user selects a desired property from the list displayed in pane 440, and uses an "Insert into Field" button 470 to insert the selected property into field 420 or field 440. Similarly, a lower right pane 480 displays a list of operators that can be used within conditions in the "Where:" clause. A user clicks on an "Insert into Field" button 490 to insert the selected operator into field 440.

Reference is now made to FIG. 5, which is a simplified block diagram of a data locator system, in accordance with a preferred embodiment of the present invention. FIG. 5 includes modeling layer 110 and mapping layer 120 from the system illustrated in FIG. 1. As in FIG. 1, FIG. 5 is also divided into three parts by dashed vertical lines; namely, data schemas and data sources on the left, central ontology model on the right, and an engine in the middle that maps items from the left to the right or vice versa.

FIG. 5 includes a data locator 530, used to identify data schema constructs corresponding to specified classes and properties of the ontology, and to identify data sources having data for such constructs.

Reference is now made to FIG. 6, which is a simplified flowchart of a method for locating data, in accordance with a preferred embodiment of the present invention. FIG. 6 includes steps 210 and 220 from the flowchart of FIG. 2. At step 630 constructs of the data schemas that map to one or more specified classes or properties of the central ontology model are identified. In a preferred embodiment of the present invention, data sources that have data for such constructs are also identified.

Reference is now made to FIG. 7, which is an illustration of a user interface for locating data within at least one data source, in accordance with a preferred embodiment of the

8

present invention. Shown in FIG. 7 is a window 700 for locating data corresponding to selected classes or properties from a central ontology model. An upper left pane 710 displays components of a project named Order. Shown highlighted in pane 710 is a "Data Sources" component. A lower left pane 720 displays a class hierarchical view of an ontology model. Pane 710 indicates that all classes inherit from a superclass named Being. Shown highlighted in pane 710 is a class OrderItem, which is the subject class for data locating.

An upper right pane 730 displays properties defined on the class OrderItem. Shown highlighted in pane 730 is a property named quantity. In a preferred embodiment, the present invention is used to generate a list of data schema constructs and data sources having data for these constructs in a lower right pane 740. For example, pane 740 indicates that Field53 of a relational database table named Orders, within a relational database schema named TRACKER, corresponds to a representation of the property quantity, specifically to a value formatted as a real number in scientific notation. Moreover a database named GEOSTRAT has data for Field53.

An important feature of the present invention is the ability to locate data schema constructs that correspond to different representations of a specified property, such as quantity, and to identify the specific formats in which the data for such constructs are represented.

More generally, in a preferred embodiment the present invention locates data schema constructs that correspond to properties of a specified property. Consider, for example, a database table, Employee, mapped to an ontology class Employees, having a column for a laptop. Such a column Employee.laptop may be mapped to a composition of ontology properties Employee.laptop.model.name.noWhiteSpace. Here laptop is a property of class Employees of type Laptop, model is a property of class Laptop of type ComputerModel, name is a property of class ComputerModel of type String, and noWhiteSpace is a property of class String of type String. It is noted that when the present invention locates data for an employee's laptop, it points to the database table column Employee.laptop. It also provides the tail of the property so as to indicate that the column Employee.laptop contains data in the form of model.name.noWhiteSpace; i.e., the name of the laptop model formatted with no white space.

Finally, a pane 750 is used for displaying system messages regarding the results of the data locating process.

The data location features of the present invention may be used for data descriptor propagation as well. Specifically, a user may designate certain classes and properties of a central ontology as being of a certain type, such as "private" or "restricted use," and in accordance with a preferred embodiment of the present invention, such type designation is propagated to data sources having data in schema constructs that correspond to the designated classes and properties. Appropriate data in fields of relational database tables and in elements of XML documents is thereby designated as being private or of restricted use.

Reference is now made to FIG. 8, which is an illustration of a visual display for displaying data location results, in accordance with a preferred embodiment of the present invention. Shown in FIG. 8 are relational database schemas and XML schemas that have constructs corresponding to an ontology class Order and its properties.

As can be seen in FIG. 8, a relational database named Logistics conforming to a relational database schema named Shipping has a table named PurchaseOrder that corresponds

to the ontology class Order. Furthermore, the table fields PurchaseOrder.id and PurchaseOrder.totalcost correspond respectively to the ontology properties Order.id and Order.total, the former being represented in integer format and the latter being represented in the format of a string in scientific notation. In addition, the table PurchaseOrder has a field with a foreign key to a table PurchaseOrderEntry, the key corresponding to the ontology property Order.customer.

As can also be seen in FIG. 8, an XML schema named orderTargetNS has a complex type named Shipment that corresponds to the ontology class Order.

The data location feature of the present invention is useful in generating location-based reports. Such reports provide strong analysis capabilities and assist in identifying data duplication, data redundancy and data inconsistency. For example, such reports may reveal that a specific database ignores a particular property, or that multiple data sources relate to the same class with each one containing data for different subsets of properties of the class.

Such reports provide statistics for classes and properties of an ontology, including inter alia the number of data sources that are mapped to them, distributed according to source type, source location, source owner and source update frequency. Such reports also show the percentage of ontology concepts covered by a given data source, including a graph and table indicating the number of properties of a given class mapped to the data source, and including a list of such properties. Such reports also indicate the number of instances per ontology class or property, and the number of the instances having only partial data.

EXAMPLES

For purposes of clarity and exposition, the workings of the present invention are described through three examples, followed by a general description of implementation. The first three examples hereinbelow correspond to the corresponding first three examples described in applicant's co-pending application U.S. Ser. No. 10/053,045, referenced hereinabove, the contents of which are hereby incorporated by reference. The fourth example hereinbelow corresponds to the seventh example described in applicant's co-pending application U.S. Ser. No. 10/053,045.

Data Query—A First Example: Schoolchildren

In a first example, three source tables are given as follows:

TABLE I

5

Source Table S ₁ for First Example		
Name	School_Attending	Mother_NI_Number

10

TABLE II

15

Source Table S ₂ for First Example			
NI_Number	Name	Region	Car_Number

20

TABLE III

25

Source Table S ₃ for First Example		
Name	Location	HeadTeacher

30

The underlying ontology is illustrated in FIG. 9. The mapping of the source schema into the ontology is as follows, where the symbol o is used to indicate composition of properties.

TABLE IV

Mapping from Source schema to Ontology for First Example		
schema	Ontology	Property Index
S ₁	Class: Child	
S ₁ .Name	Property: Child.name	6
S ₁ .School_Attending	Property: Child.school_attending.name	10o9
S ₁ .Mother_NI_Number	Property: Child.mother.national_insurance_number	4o5
S ₂	Class: Person	
S ₂ .NI_Number	Property: Person.national_insurance_number	4
S ₂ .Name	Property: Person.name	3
S ₂ .Region	Property: Person.region_of_residence	1
S ₂ .Car_Number	Property: Person.car_registration_number	2
S ₃	Class: School	
S ₃ .Name	Property: School.name	10
S ₃ .Location	Property: School.location	12
S ₃ .HeadTeacher	Property: School.headteacher.nameheadteacher	3o11

55 The following ontology language query is used to request a list of children's names and their mothers' names, for all children in the database who attend schools in London.

60 SELECT: Child.name, Child.mother.name
 FROM: Child
 WHERE: Child.school_attending.location="London"

65 Using the present invention, the above ontology language query is converted to the following corresponding SQL query:

11

```

SELECT
  S1.Name, S2.Name
FROM
  S1, S2, S3
WHERE
  S2.NL_Number = S1.Mother__NL_Number AND
  S3.Name = S1.School__Attending AND
  S3.Location = "London"
    
```

Data Query—A Second Example: Employees

In a second example, four source tables are given as follows:

12

TABLE VII

Source Table S ₁ for Second Example		
ID #	Room_Assignment	Telephone #

5

TABLE VIII

Source Table S ₄ for Second Example	
Department	Budget

10

15

The underlying ontology is illustrated in FIG. 10. The mapping of the source schema into the ontology is as follows:

TABLE IX

Mapping from Source schema to Ontology for Second Example		
schema	Ontology	Property Index
S ₁	Class: Employee	
S ₁ .Emp_ID #	Property: Employee.ID #	4
S ₁ .Name	Property: Employee.name	3
S ₁ .Department	Property: Employee.departmental__affiliation.code	807
S ₂	Class: Employee	
S ₂ .Employee__Name	Property: Employee.name	3
S ₂ .Supervisor	Property: Employee.supervisor.name	306
S ₂ .Project	Property: Employee.project__assignment	5
S ₃	Class: Employee	
S ₃ .ID #	Property: Employee.ID #	4
S ₃ .Room_Assignment	Property: Employee.room__number	1
S ₃ .Telephone #	Property: Employee.tel #	2
S ₄	Class: Department	
S ₄ .Department	Property: Department.code	8
S ₄ .Budget	Property: Department.budget__amount	9

TABLE V

Source Table S ₁ for Second Example		
Emp_ID #	Name	Department

The following ontology language query is used to request a list of employees' names, departments and supervisors, for all employees in the database who are assigned to Room 101.

SELECT: Employee.name, Employee.department, Employee.supervisor

FROM: Employee

WHERE: Employee.room_number="101"

Using the present invention, the above ontology language query is converted to the following corresponding SQL query:

TABLE VI

Source Table S ₂ for Second Example		
Employee__Name	Supervisor	Project

55

```

SELECT
  S1.Name, S1.Department, S2.Supervisor
FROM
    
```

60

```

  S1, S2, S3
WHERE
  S2.Employee__Name = S1.Name AND
  S3.ID # = S1.Emp_ID # AND
  S3.Room_Assignment = "101"
    
```

65

13

Data Query—A Third Example: Airline Flights

In a third example, two source tables are given as follows:

TABLE X

Source Table S ₁ for Third Example		
Index	APName	Location

TABLE XI

Source Table S ₂ for Third Example		
FlightID	FromAirport	ToAirport

The underlying ontology is illustrated in FIG. 11. The mapping of the source schema into the ontology is as follows:

TABLE XII

Mapping from Source schema to Ontology for Third Example			
schema	Ontology	Property	Index
S ₁	Class: Airport		
S ₁ .Index	Property: Airport.index		3
S ₁ .APName	Property: Airport.name		1
S ₁ .Location	Property: Airport.location		2
S ₂	Class: Flight		
S ₂ .FlightID	Property: Flight.ID #		6
S ₂ .FromAirport	Property: Flight.from_ airport.name		1o4
S ₂ .ToAirport	Property: Flight.to_ airport.name		1o5

The following ontology language query is used to request a list of flight numbers, for all flights from New York to San Francisco.

```
SELECT: Flight.ID#
FROM: Flight
WHERE: Flight.from_ airport.location="New York" AND
Flight.to_ airport.location="San Francisco"
```

Using the present invention, the above ontology language query is converted to the following corresponding SQL query:

```
SELECT
FROM
WHERE
    S2.FlightID,
    S1 S11, S1 S12, S2
    S11.APName = S2.FromAirport AND
    S12.APName = S2.ToAirport AND
    S11.Location = "New York" AND
    S12.Location = "San Francisco"
```

Data Location—A Fourth Example: Employees

In the fourth example, four source tables are given as follows:

14

TABLE XIII

Source Table S ₁ for Seventh Example	
ID	Group

TABLE XIV

Source Table S ₂ for Seventh Example	
ID	Email

TABLE XV

Source Table S ₃ for Seventh Example	
ID	Email_Address

TABLE XVI

Source Table S ₄ for Seventh Example	
ID	Unit

The underlying ontology is illustrated in FIG. 12. The mapping of the source schema into the ontology is as follows:

TABLE XVII

Mapping from Source schema to Ontology for Seventh Example			
schema	Ontology	Property	Index
S ₁	Class: Employee		
S ₁ .ID	Property: Employee.ID #		2
S ₁ .Group	Property: Employee.department		4
S ₂	Class: Employee		
S ₂ .ID	Property: Employee.ID #		2
S ₂ .Email	Property: Employee.e-mail		3
S ₃	Class: Employee		
S ₃ .ID	Property: Employee.ID #		2
S ₃ .Email_Address	Property: Employee.e-mail		3
S ₄	Class: Employee		
S ₄ .ID	Property: Employee.ID #		2
S ₄ .Unit	Property: Employee.department		4

For the property Employee.e-mail, the present invention indicates that this data is located at S₂.Email and S₄.Email_Address. Similarly, for the property Employee.department, the present invention indicates that this data is located at S₁.Group and S₃.Unit.

Multi-Valued Properties in an Ontology

A multi-valued property is a property that maps an instance in its source class to more than one instance in its target class.

For example, a property `Person.children` may map a person from class `Person` to more than one child in class `Person`.

In order to effectively allow for multi-valued properties in addition to single-valued properties, provision is made in an ontology for “container” classes. Generally, container classes are built from simpler classes using tags for set, list and bag. The set symbol is used for describing a class comprising subsets of a class. The list symbol is used for describing a class comprising ordered subsets of a class; namely, sequences. The bag symbol is used for describing unordered finite sequences of a class, namely, subsets that can contain repeated elements. Thus `set[C]` describes the class of sets of instances of a class `C`, `list[C]` describes the class of lists of instances of class `C`, and `bag[C]` describes the class of bags of instances of class `C`. Elements of a sequence `se list[C]` are denoted by `s[0]`, `s[1]`, . . .

In terms of formal mathematics, for a set `C`, `set[C]` is 2^C , the power set of `C`; `bag[C]` is N^C , where N is the set of non-negative integers; and `list[C]` is C^N ; more precisely, the subset of C^N consisting of finite sequences. There are natural projections

$$\text{list}[C] \xrightarrow{\phi} \text{bag}[C] \xrightarrow{\psi} \text{set}[C]. \tag{1}$$

Specifically, for a sequence $(c[0], c[1], \dots, c[n]) \in \text{list}[C]$, $\phi(c[0], c[1], \dots, c[n])$ is the element `febag[C]` that is the “frequency histogram” defined by $f(c) = \#\{0 \leq i \leq n: c[i] = c\}$; and for `febag[C]`, $\psi(f) \in \text{set}[C]$ is the subset of `C` given by the support of `f`, namely, $\text{supp}(f) = \{c \in C: f(c) > 0\}$. It is noted that the composite mapping $\phi \circ \psi$ maps the sequence $(c[0], c[1], \dots, c[n])$ into the set of its elements $\{c[0], c[1], \dots, c[n]\}$. Each projection in Equation (1) removes some structure. Specifically, ϕ removes order and ψ removes repetition.

Using container sets, provision is made for multi-valued properties. Generally, the target of a multi-valued property is a container class. For example, a multi-valued property named `children` may have a class `Person` as its source and a container class `set[Person]` as its target.

A single-valued property $p: C \rightarrow D$, from a class `C` to a class `D` has natural liftings to `set[C]`, `bag[C]` and `list[C]`. Specifically, the lifting to sets, $p: \text{set}[C] \rightarrow \text{set}[D]$ is defined by $p(S) = \{p(s): s \in S\}$ for subsets $S \rightarrow C$. The lifting to bags, $p: \text{bag}[C] \rightarrow \text{bag}[D]$ is defined by $p(f): d \mapsto$

$$p(f): d \mapsto \sum_{c \in p^{-1}(d)} f(c),$$

for $f \in N^C$. The lifting to lists, $p: \text{list}[C] \rightarrow \text{list}[D]$ is defined by $p(s) = \text{pos}$, for a sequence $s \in C^N$.

An important feature of these liftings of `p` is their consistency with the projections from Equation (1), as illustrated by the commutative diagrams in FIG. 13.

Compositions of multi-valued properties is governed by Equation (1). Specifically, a multi-valued property `p` with target `bag[C]` can be followed by a property `q` with source `set[C]` according to $qp = q \circ \psi \circ \phi$. Similarly, a multi-valued property `p` with target `list[C]` can be followed by a property `q` with source `bag[C]` according to $qp = q \circ \phi \circ \psi$, and a multi-valued property `p` with target `list[C]` can be followed by a property `q` with source `set[C]` according to $qp = q \circ \phi \circ \psi \circ \phi$.

For example, a property `totalCost: bag[Components] → Values` can be applied to `list[Components]` by converting a list to a bag via the projection ϕ .

Familiar algebraic data structures including inter alia vectors and matrices correspond to lists of pre-determined length.

For data models that require the notion of an ordered set of instances without repetition, such a structures are considered as a special subset of `list[C]`; namely, lists without repetition.

To simplify compositions of properties, in one embodiment of the present invention special identifications are made to collapse containers of containers. Specifically, `set[set[C]]` is identified as `set[C]`, through the set union. Thus, `Person.children.children` is identified as a set of people corresponding to a person’s grandchildren, instead of as a set of sets of people. Similarly, `bag[bag[C]]` is identified as `bag[C]` by adding frequencies, and `Ust[list[C]]` is identified as `list[C]` by combining a list of lists into a single list.

Optionally, `set[bag[C]]`, `bag[set[C]]`, `set[list[C]]` and `list[set[C]` may be identified as `set[C]` by removing order and repetition structure through use of the projections ϕ and ψ from Equation (1). Similarly, `bag[list[C]]` and `list[bag[C]]` may be identified as `bag[C]` by removing order structure through use of the projection ϕ from Equation (1).

In an alternate embodiment of the present invention, containers of containers are not collapsed.

In data models it is often important to indicate minimum and maximum numbers of elements in the target of a multi-valued property. For example, a property `Person.parents` has a maximum of two elements in its target set. To this end, multi-valued properties may have `min_elements` and `max_elements` as attributes. Such attributes serve to limit the target of multi-valued properties to specific subsets of `set[C]`, `bag[C]` and `list[C]`; namely, those subsets, bags and lists, respectively, having a number of elements, `n`, satisfying $\text{min_elements} \leq n \leq \text{max_elements}$.

In a preferred embodiment of the present invention, constraints may be imposed relating single-valued and multi-valued properties. For example, a constraint

```
BillOfGoods.totalCost = Σ BillOfGoods.items[k].cost
relates properties totalCost:
BillOfGoods → Values, items: BillOfGoods → list
[Parts] and cost: Parts → Values.
```

Multi-valued properties can be used to model n-ary relationships within an ontology. For example, a tertiary relationship $R \subset C_1 \times C_2 \times C_3$ can be modeled as a property $r: C_1 \rightarrow \text{set}[C_2 \times C_3]$, defined according to $r(c_1) = \{(c_2, c_3): R(c_1, c_2, c_3)\}$. The ability to model relationships enables the present invention to import data models that support relationships, such as the entity-relationship data model. Many-to-many relationships can be imported as multi-valued ontology properties. A reference for the entity-relationship data model is Garcia-Molina, Ullman, Jeffrey D. and Widom, Jennifer, “Database Systems: The Complete Book,” Prentice Hall, Upper Saddle River, N.J., 2002.

In a preferred embodiment of the present invention, multi-valued properties are used for embedding data schema into an ontology model, for schema that include constructs that can include more than one data elements.

For example, XML complexTypes can include sets of sequences of XML elements therewithin. Consider the XML schema from Example Fourteen of applicant’s above-referenced co-pending application U.S. Ser. No. 10/053,045.


```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="storage" type="Storage"/>
  <xs:complexType name="Storage">
    <xs:sequence>
      <xs:element name="articles" type="Documents"/>
      <xs:element name="reviews" type="Documents"/>
      <xs:element name="letters" type="Letters"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="Documents">
    <xs:sequence>
      <xs:element name="document" type="Document" minOccurs="0"
        maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="Letters">
    <xs:sequence>
      <xs:element name="letter" type="Letter" minOccurs="0"
        maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="Document">
    <xs:sequence>
      <xs:element name="author" type="xs:string" minOccurs="0"
        maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="title"/>
  </xs:complexType>
  <xs:complexType name="Letter">
    <xs:sequence>
      <xs:element name="author" type="xs:string" minOccurs="0"
        maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="name"/>
    <xs:attribute name="subject"/>
    <xs:attribute name="receiver"/>
  </xs:complexType>
</xs:schema>

```

The underlying ontology is illustrated in FIG. 14. Observe that the XML element “letters” within the XML complex-Type “storage” corresponds to the multi-valued property Storage.letters from the class Storage to the container class set[Letter]. Similarly the XML elements “articles” and “reviews” within the XML complexType “storage” correspond to multi-valued properties Storage.articles and Storage.reviews, respectively, from the class Storage to the container class set[Document].

Cobol Copy Books

The present invention can be used to query COBOL records. Using COBOL, variables are declared as follows:

01 StudentDetails.		
02 StudentId	PIC 9(7).	
02 StudentName.		
03 FirstName	PIC X(10).	
03 MiddleInitial	PIC X.	
03 Surname	PIC X(15).	
02 DateofBirth.		
03 DayofBirth	PIC 99.	
03 MonthofBirth	PIC 99.	
03 YearofBirth	PIC 9(4).	
02 CourseCode	PIC X(4).	

The variable StudentDetails includes group items, emphasized above in bold, and elementary items. Data can be read or written from variables provided that the size of the data corresponds to the declared sizes.

In the above typing for elementary items, 9 denotes a number and X denotes an alphanumeric. The notation 9(5) denotes a five digit non-negative integer. In addition, V is used to denote a decimal point, and S is used to denote a sign.

A COBOL Copy Book (CCB) is the metadata for COBOL format data that is saved to disk. A CCB defines the format of a COBOL record. The instances of a record defined by a CCB are in binary format.

It is noted that the above CCB corresponds to the following XML schema:

```

50 <element name="StudentDetails">
  <complexType>
    <sequence>
      <element name="StudentId" type="integer"/>
      <element name="StudentName">
        <complexType>
          <sequence>
            <element name="FirstName" type="string"/>
            <element name="MiddleInitial" type="string"/>
            <element name="Surname" type="string"/>
          </sequence>
        </complexType>
      </element>
      <element name="DateofBirth">
        <complexType>
          <sequence>
            <element name="DayofBirth" type="integer"/>
            <element name="MonthofBirth" type="integer"/>
            <element name="YearofBirth" type="integer"/>
          </sequence>
        </complexType>
    </sequence>
  </complexType>

```

65

-continued

```

</element>
<element name="CourseCode" type="string"/>
</sequence>
</complexType>
</element>

```

Specifically, a CCB corresponds to a restricted type of XML schema. Group items in a CCB correspond to complex-Types in an XML schema, and elementary items in a CCB correspond to elements in an XML schema.

Implementation Details

The present invention overcomes several challenges when converting queries from the three-clause SELECT-FROM-WHERE ontology query language to data schema query language. For conversion to SQL, these include:

1. Joining appropriate tables corresponding to one or more classes in the FROM clause.
2. Identifying fields corresponding to one or more properties in the SELECT clause.
3. Converting conditions in the WHERE clause to SQL format.

The former two challenges involve the mapping from tables and fields of relational database schemas into classes and properties of an ontology model, and can be overcome using applicant's technology for generating SQL transformations, as described in applicant's co-pending application U.S. Ser. No. 10/053,045, referenced hereinabove. Specifically, this application describes generating transformations for populating target data conforming to a target data schema from source data conforming to a source data schema. In particular, this application describes how to express target schema constructs in terms of source schema constructs. For the case at hand, the target schema constructs are the properties in the SELECT clause.

The latter challenge involves converting a logical expression from ontology format into SQL format, and can be overcome using Codd's normalization for converting from first order logic to SQL.

For conversion of an ontology query to an XQuery, three similar challenges arise, where complex types correspond to one or more classes in the FROM clause and elements and attributes correspond to one or more properties in the SELECT clause. As above, applicant's co-pending U.S. Ser. No. 10/053,045 describes generating XSL transformations, which can be used to overcome the first two challenges.

The latter challenge involves converting a logical expression from ontology format to XQuery format.

In the foregoing specification, the invention has been described with reference to specific exemplary embodiments thereof. It will, however, be evident that various modifications and changes may be made to the specific exemplary embodiments without departing from the broader spirit and scope of the invention as set forth in the appended claims.

Accordingly, the specification and drawings are to be regarded in an illustrative rather than a restrictive sense.

What is claimed is:

1. A computer readable medium containing a COBOL mapping system for embedding a COBOL Copy Book including group items and elementary items into an ontology model including classes and properties, the COBOL mapping system comprising:
 - a first data construct usable to map at least one COBOL group item within a COBOL Copy Book to at least one class of an ontology model;
 - a second data construct usable to map at least one COBOL elementary item within the COBOL Copy Book to at least one of: at least one property of the ontology model or a composition of properties of the ontology model; and
 - a third data construct comprising a first query referring to the ontology model, wherein the first query is expressed in an ontology language; and
 - a fourth data construct comprising a second query referring to a data schema embedded within the ontology model, wherein the first data construct and the second data construct were used to convert the first query into the second query, wherein the data schema relates to the at least one COBOL group item, and wherein the second query is expressed in a data schema language associated with COBOL.
2. A computer-implemented method for embedding a COBOL Copy Book including group items and elementary items into an ontology model including classes and properties, the computer-implemented comprising:
 - mapping at least one COBOL group item within a COBOL Copy Book to at least one class of an ontology model, wherein a first mapping is formed;
 - mapping, using at least one data construct, at least one COBOL elementary item within the COBOL Copy Book to at least one property of the ontology model or a composition of properties of the ontology model, wherein a second mapping is formed;
 - storing the first mapping and the second mapping; and
 - using the first mapping and the second mapping to convert a first query referring to the ontology model into a corresponding second query referring to a data schema embedded within the ontology model, wherein the data schema relates to the at least one COBOL group item, wherein the first query is expressed in an ontology language, and wherein the second query is expressed in a data schema language associated with COBOL.
3. The computer readable medium of claim 1 further comprising:
 - a fifth data structure comprising a result of execution of the second query.
4. The method of claim 2 further comprising:
 - executing the second query.

* * * * *