

(19) United States (12) Patent Application Publication (10) Pub. No.: US 2004/0216030 A1

Oct. 28, 2004 (43) Pub. Date:

Hellman et al.

(54) METHOD AND SYSTEM FOR DERIVING A TRANSFORMATION BY REFERRING SCHEMA TO A CENTRAL MODEL

(76) Inventors: Ziv Z. Hellman, Jerusalem (IL); Marcel Zvi Schreiber, Jerusalem (IL); Tom Y Yuval, Jerusalem (IL)

> Correspondence Address: **BLAKELY SOKOLOFF TAYLOR & ZAFMAN 12400 WILSHIRE BOULEVARD** SEVENTH FLOOR LOS ANGELES, CA 90025-1030 (US)

- (21) Appl. No.: 10/053,045
- (22) Filed: Jan. 15, 2002

Related U.S. Application Data

(63) Continuation-in-part of application No. 09/866,101, filed on May 25, 2001.

Publication Classification

- (51) Int. Cl.⁷ G06F 15/00

ABSTRACT (57)

A method for transforming data from one data schema to another including receiving a source data schema and a target data schema, mapping the source data schema into an ontology model, mapping the target data schema into the ontology model, and deriving a transformation for transforming data conforming to the source data schema into data conforming to the target data schema, using the ontology model. A system is also described and claimed.

























FIG. 9B







FIG. 9E







FIG. 11B









elementFormDefault="qualified" attributeFormDrexs: element name="reservation" type="Reser - <xs: complextype="" name="Filghts"> - <xs: sequence=""> <xs: element="" m<="" name="Fight" th="" type="Fiight"><th> </th></xs:></xs:></xs:>	
maxOccurs="unbounded" /> - <xs: complex="" name#"elight"="" type=""> - <xs: sequence=""></xs:></xs:>	type="FlightReservation" /> - <xs:complextype name="FlightInfo"> - <xs:sequence> <xs:element name="securityLeve!" type="xs:string"></xs:element> <xs:element <="" name="flightOccurence" th="" type="Journey"></xs:element></xs:sequence></xs:complextype>
<pre><ks:element name='atrline"' typ<br="" type="
<ks:element name= departureAirport"><ks:element name='distanceInKm"' pre="" stert"<="" sterttime"="" type="ks:element name"></ks:element></ks:element></pre>	minOccurs="0" maxOccurs="unbounded" /> - <xs:complextype name="Journey"> - <xs:sequence></xs:sequence></xs:complextype>
<pre> <!--</td--><td><pre><xs:element name="airline" type="Company"></xs:element> <xs:element name="arrival" type="Airport"></xs:element> <xs:element name="departure" type="Airport"></xs:element> <xs:element name="distanceInMiles" type="xs:float"></xs:element> <xs:element name="id" type="xs:string"></xs:element> </pre></td></pre>	<pre><xs:element name="airline" type="Company"></xs:element> <xs:element name="arrival" type="Airport"></xs:element> <xs:element name="departure" type="Airport"></xs:element> <xs:element name="distanceInMiles" type="xs:float"></xs:element> <xs:element name="id" type="xs:string"></xs:element> </pre>
 <ks: cdmpiex="" tyge=""> #</ks:> <ks: cdmpiex="" tyge=""> #</ks:> <ks: a<="" cets="" li="" section=""> <kg: <="" element="" li="" namfe="flights" type="flights"> </kg:></ks:> <ks: element="" flightreservation"="" name="passenger" type="Passes </td><td><pre></www.complexType> </www.complexType> </www.complexType="> </ks:>	
<pre> </pre> Swiss Air Schema FIG. 11F	<pre><ks:element name="flight" type="Journey"></ks:element> British Airways Scheme</pre>











FIG. 11J






































FIG. 23









METHOD AND SYSTEM FOR DERIVING A TRANSFORMATION BY REFERRING SCHEMA TO A CENTRAL MODEL

CROSS REFERENCES TO RELATED APPLICATIONS

[0001] This application is a continuation-in-part of assignee's pending application U.S. Ser. No. 09/866,101 filed on May 25, 2001, entitled "Method and System for Collaborative Ontology Modeling."

FIELD OF THE INVENTION

[0002] The present invention relates to data schema, and in particular to deriving transformations for transforming data from one schema to another.

BACKGROUND OF THE INVENTION

[0003] Ontology is a philosophy of what exists. In computer science ontology is used to model entities of the real world and the relations between them, so as to create common dictionaries for their discussion. Basic concepts of ontology include (i) classes of instances/things, and (ii) relations between the classes, as described hereinbelow. Ontology provides a vocabulary for talking about things that exist.

[0004] Instances/Things

[0005] There are many kinds of "things" in the world. There are physical things like a car, person, boat, screw and transistor. There are other kinds of things which are not physically connected items or not even physical at all, but may nevertheless be defined. A company, for example, is a largely imaginative thing the only physical manifestation of which is its appearance in a list at a registrar of companies. A company may own and employ. It has a defined beginning and end to its life.

[0006] Other things can be more abstract such as the Homo Sapiens species, which is a concept that does not have a beginning and end as such even if its members do.

[0007] Ontological models are used to talk about "things." An important vocabulary tool is "relations" between things. An ontology model itself does not include the "things," but introduces class and property symbols which can then be used as a vocabulary for talking about and classifying things.

[0008] Properties

[0009] Properties are specific associations of things with other things. Properties include:

- **[0010]** Relations between things that are part of each other, for example, between a PC and its flat panel screen;
- [0011] Relations between things that are related through a process such as the process of creating the things, for example, a book and its author;
- **[0012]** Relations between things and their measures, for example, a thing and its weight.

[0013] Some properties also relate things to fundamental concepts such as natural numbers or strings of characters—for example, the value of a weight in kilograms, or the name of a person.

[0014] Properties play a dual role in ontology. On the one hand, individual things are referenced by way of properties, for example, a person by his name, or a book by its title and author. On the other hand, knowledge being shared is often a property of things, too. A thing can be specified by way of some of its properties, in order to query for the values of other of its properties.

[0015] Classes

[0016] Not all properties are relevant to all things. It is convenient to discuss the source of a property as a "class" of things, also referred to as a frame or, for end-user purposes, as a category. Often sources of several properties coincide, for example, the class Book is the source for both Author and ISBN Number properties.

[0017] There is flexibility in the granularity to which classes are defined. Cars is a class. Fiat Cars can also be a class, with a restricted value of a manufacturer property. It may be unnecessary to address this class, however, since Fiat cars may not have special properties of interest that are not common to other cars. In principle, one can define classes as granular as an individual car unit, although an objective of ontology is to define classes that have important properties.

[0018] Abstract concepts such as measures, as well as media such as a body of water which cannot maintain its identity after coming into contact with other bodies of water, may be modeled as classes with a quantity property mapping them to real numbers.

[0019] In a typical mathematical model, a basic ontology comprises:

- **[0020]** A set C, the elements of which are called "class symbols;"
- [0021] For each C \in C, a plain language definition of the class C;
- [0022] A set P, the elements of which are called "property symbols;"
- [0023] For each $P \in F$:
- [0024] a plain language definition of P;
- [0025] a class symbol called the source of P; and
- [0026] a class symbol called the target of P; and
- [0027] A binary transitive reflexive anti-symmetric relation, I, called the inheritance relation on $C \times C$.

[0028] In the ensuing discussion, the terms "class" and "class symbol" are used interchangeably, for purposes of convenience and clarity. Similarly, the terms "property" and "property symbol" are also used interchangeably.

[0029] It is apparent to those skilled in the art that if an ontology model is extended to include sets in a class, then a classical mathematical relation on C×D can be considered as a property from C to sets in D.

[0030] If $I(C_1, C_2)$ then C_1 is referred to as a subclass of C_2 , and C_2 is referred to as a superclass of C_1 . Also, C_1 is said to inherit from C_2 .

[0031] A distinguished universal class "Being" is typically postulated to be a superclass of all classes in C.

- [0032] Variations on an ontology model may include:
 - **[0033]** Restrictions of properties to unary properties, these being the most commonly used properties;
 - **[0034]** The ability to specify more about properties, such as multiplicity and invertibility.

[0035] The notion of a class symbol is conceptual, in that it describes a generic genus for an entire species such as Books, Cars, Companies and People.

[0036] Specific instances of the species within the genus are referred to as "instances" of the class. Thus "Gone with the Wind" is an instance of a class for books, and "IBM" is an instance of a class for companies. Similarly, the notions of a property symbol is conceptual, in that it serves as a template for actual properties that operate on instances of classes.

[0037] Class symbols and property symbols are similar to object-oriented classes in computer programming, such as C++ classes. Classes, along with their members and field variables, defined within a header file, serve as templates for specific class instances used by a programmer. A compiler uses header files to allocate memory for, and enables a programmer to use instances of classes. Thus a header file can declare a rectangle class with members left, right, top and bottom. The declarations in the header file do not instantiate actual "rectangle objects," but serve as templates for rectangles instantiated in a program. Similarly, classes of an ontology serve as templates for instances thereof.

[0038] There is, however, a distinction between C++ classes and ontology classes. In programming, classes are templates and they are instantiated to create programming objects. In ontology, classes document common structure but the instances exist in the real world and are not created through the class.

[0039] Ontology provides a vocabulary for speaking about instances, even before the instances themselves are identified. A class Book is used to say that an instance "is a Book." A property Author allows one to create clauses "author of" about an instance. A property Siblings allows one to create statements "are siblings" about instances. Inheritance is used to say, for example, that "every Book is a PublishedWork". Thus all vocabulary appropriate to PublishedWork can be used for Book.

[0040] Once an ontology model is available to provide a vocabulary for talking about instances, the instances themselves can be fit into the vocabulary. For each class symbol, C, all instances which satisfy "is a C" are taken to be the set of instances of C, and this set is denoted B(C). Sets of instances are consistent with inheritance, so that $B(C_1) \subset B(C_2)$ whenever C_1 is a subclass of C_2 . Property symbols with source $B(C_1)$ and target $B(C_2)$. It is noted that if class C_1 inherits from class C, then every instance of C_1 is also an instance of C, and it is therefore known already at the ontology stage that the vocabulary of C is applicable to C_1 .

[0041] Ontology enables creation of a model of multiple classes and a graph of properties therebetween. When a class is defined, its properties are described using handles to related classes. These can in turn be used to look up properties of the related classes, and thus properties of properties can be accessed to any depth.

[0042] Provision is made for both classes and complex classes. Generally, complex classes are built up from simpler classes using tags for symbols such as intersection, Cartesian product, set, list and bag. The "intersection" tag is followed by a list of classes or complex classes. The "Cartesian product" tag is also followed by a list of classes or complex classes. The set symbol is used for describing a class comprising subsets of a class, and is followed by a single class or complex class. The list symbol is used for describing a class comprising ordered subsets of a class; namely, finite sequences, and is followed by a single class or complex class. The bag symbol is used for describing unordered finite sequences of a class, namely, subsets that can contain repeated elements, and is followed by a single class or complex class. 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.

[0043] In terms of formal mathematics, for a set S, set[S] is P(S), the power set of S; bag[S] is N^S, where N is the set of non-negative integers; and list[S] is



[0044] There are natural mappings

$$\operatorname{list}[S] \xrightarrow{\phi} \operatorname{bag}[S] \xrightarrow{\psi} \operatorname{set}[S]. \tag{1}$$

[0045] Specifically, for a sequence $(s_1, s_2, \ldots, s_n) \epsilon$ list[S], $\phi(s_1, s_2, \ldots, s_n)$ is the element $f\epsilon bag[S]$ that is the "frequency histogram" defined by $f(s)=\#\{1 \le i \le n: s_i = s\}$; and for $f\epsilon bag[S]$, $\psi(f)Eset[S]$ is the subset of S given by the support of f, namely, $supp(f)=\{s\epsilon S: f(s)>0\}$. It is noted that the composite mapping $\phi\psi$ maps a the sequence (s_1, s_2, \ldots, s_n) into the set of its elements $\{s_1, s_2, \ldots, s_n\}$. For finite sets S, set[S] is also finite, and bag[S] and list[S] are countably infinite.

[0046] A general reference on ontology systems is Sowa, John F., "Knowledge Representation," Brooks/Cole, Pacific Grove, Calif., 2000.

[0047] Relational database schema (RDBS) are used to define templates for organizing data into tables and fields. SQL queries are used to populate tables from existing tables, generally by using table join operations. Extensible markup language (XML) schema are used to described documents for organizing data into a hierarchy of elements and attributes. XSLT script is used to generate XML documents from existing documents, generally by importing data between tags in the existing documents. XSLT was originally developed in order to generate HTML pages from XML documents.

[0048] A general reference on relation databases and SQL is the document "Oracle 9i: SQL Reference," available on-line at http://www.oracle.com. XML, XML schema, XPath and XSLT are standards of the World-Wide Web Consortium, and are available on-line at http://www.w3.org.

[0049] Often multiple schema exist for the same source of data, and as such the data cannot readily be imported or exported from one application to another. For example, two airline companies may each run applications that process relational databases, but if the relational databases used by the two companies conform to two different schema, then neither of the companies can readily use the databases of the other company. In order for the companies to share data, it is necessary to export the databases from one schema to another.

[0050] There is thus a need for a tool that can transform data conforming with a first schema into data that conforms with a second schema.

SUMMARY OF THE INVENTION

[0051] The present invention provides a method and system for deriving transformations for transforming data from one schema to another. The present invention describes a general method and system for transforming data confirming with an input, or source data schema into an output, or target data schema. In a preferred embodiment, the present invention can be used to provide (i) an SQL query, which when applied to relational databases from a source RDBS, populates relational databases in a target RDBS; and (ii) XSLT script which, when applied to documents conforming with a source XML schema generates documents conforming with a target XML schema.

[0052] The present invention preferably uses an ontology model to determine a transformation that accomplishes a desired source to target transformation. Specifically, the present invention employs a common ontology model into which both the source data schema and target data schema can be mapped. By mapping the source and target data schema into a common ontology model, the present invention derives interrelationships among their components, and uses the interrelationships to determine a suitable transformation for transforming data conforming with the source data schema.

[0053] Given a source RDBS and a target RDBS, in a preferred embodiment of the present invention an appropriate transformation of source to target databases is generated by:

- [0054] (i) mapping the source and target RDBS into a common ontology model;
- [0055] (ii) representing table columns of the source and target RDBS in terms of properties of the ontology model;
- [0056] (iii) deriving expressions for target table columns in terms of source table columns; and
- [0057] (iv) converting the expressions into one or more SQL queries.

[0058] Although the source and target RDBS are mapped into a common ontology model, the derived transformations of the present invention go directly from source RDBS to target RDBS without having to transform data via an ontological format. In distinction, prior art Universal Data Model approaches transform via a neutral model or common business objects. **[0059]** The present invention applies to N relational database schema, where $N \ge 2$. Using the present invention, by mapping the RDBS into a common ontology model, data can be moved from any one of the RDBS to any other one. In distinction to prior art approaches that require on the order of N^2 mappings, the present invention requires at most N mappings.

[0060] For enterprise applications, SQL queries generated by the present invention are preferably deployed within an Enterprise Application Integration infrastructure. Those skilled in the art will appreciate that transformation languages other than SQL that are used by enterprise application infrastructures can be generated using the present invention. For example, IBM's ESQL language can similarly be derived for deployment on their WebSphere MQ family of products.

[0061] Given a source XML schema and a target XML schema, in a preferred embodiment of the present invention an appropriate transformation of source to target XML documents is generated by:

- [0062] (i) mapping the source and target XML schema into a common ontology model;
- **[0063]** (ii) representing elements and attributes of the source and target XML schema in terms of properties of the ontology model;
- [0064] (iii) deriving expressions for target XML elements and XML attributes in terms of source XML elements and XML attributes; and
- [0065] (iv) converting the expressions into an XSLT script.

[0066] There is thus provided in accordance with a preferred embodiment of the present invention a method for deriving transformations for transforming data from one data schema to another, including receiving a source data schema and a target data schema, mapping the source data schema into an ontology model, mapping the target data schema into the ontology model, and deriving a transformation for transforming data conforming to the source data schema into data conforming to the target data schema, using the ontology model.

[0067] There is further provided in accordance with a preferred embodiment of the present invention a system for deriving transformations for transforming data from one data schema to another, including a schema receiver receiving a source data schema and a target data schema, a mapping processor mapping a data schema into an ontology model, and a transformation processor deriving a transformation for transforming to the source data schema, based on respective source and target mappings generated by said mapping processor for mapping said source data schema and said target data schema into a common ontology model.

[0068] There is yet further provided in accordance with a preferred embodiment of the present invention a method for building an ontology model into which data schema can be embedded, including receiving at least one data schema, and building an ontology model into which the at least one data schema can be embedded.

[0069] There is additionally provided in accordance with a preferred embodiment of the present invention a system for building an ontology model into which data schema can be embedded, including a schema receiver receiving at least one data schema, and a model builder building an ontology model into which the at least one data schema can be embedded.

[0070] There is moreover provided in accordance with a preferred embodiment of the present invention an article of manufacture including one or more computer-readable media that embody a program of instructions for transforming data from one schema to another, wherein the program of instructions, when executed by a processing system, causes the processing system to receive a source data schema and a target data schema, map the source data schema into an ontology model, map the target data schema into data conforming to the source data schema into data conforming to the target relational database schema, using the ontology model.

[0071] There is further provided in accordance with a preferred embodiment of the present invention an article of manufacture including one or more computer-readable media that embody a program of instructions for building a common ontology model into which data schema can be embedded, wherein the program of instructions, when executed by a processing system, causes the processing system to receive at least one data schema, and build an ontology model into which the at least one data schema can be embedded.

BRIEF DESCRIPTION OF THE DRAWINGS

[0072] The present invention will be more fully understood and appreciated from the following detailed description, taken in conjunction with the drawings in which:

[0073] FIG. 1 is a simplified flowchart of a method for deriving transformations for transforming data from one schema to another, in accordance with a preferred embodiment of the present invention;

[0074] FIG. 2 is a simplified block diagram of a system for deriving transformations for transforming data from one schema to another, in accordance with a preferred embodiment of the present invention;

[0075] FIG. 3 is a simplified flowchart of a method for building a common ontology model into which one or more data schema can be embedded, in accordance with a preferred embodiment of the present invention;

[0076] FIG. 4 is a simplified block diagram of a system for building a common ontology model into which one or more data schema can be embedded, in accordance with a preferred embodiment of the present invention;

[0077] FIG. 5 is a simplified illustration of a mapping from an RDBS into an ontology model, in accordance with a preferred embodiment of the present invention;

[0078] FIG. 6 is a second simplified illustration of a mapping from an RDBS into an ontology model, in accordance with a preferred embodiment of the present invention;

[0079] FIG. 7 is a simplified illustration of relational database transformations involving constraints and joins, in accordance with a preferred embodiment of the present invention;

[0080] FIG. 8 is a simplified illustration of use of a preferred embodiment of the present invention to deploy XSLT scripts within an EAI product such as Tibco;

[0081] FIGS. 9A-9E are illustrations of a user interface for a software application that transforms data from one relational database schema to another, in accordance with a preferred embodiment of the present invention;

[0082] FIG. 10 is an illustration of a user interface for an application that imports an RDBS into the software application illustrated in **FIGS. 8A-8E**, in accordance with a preferred embodiment of the present invention;

[0083] FIGS. 11A-11R are illustrations of a user interface for a software application that transforms data from one XML schema to another, in accordance with a preferred embodiment of the present invention;

[0084] FIG. 12 is an illustration of ontology model corresponding to a first example;

[0085] FIG. 13 is an illustration of ontology model corresponding to a second example;

[0086] FIG. 14 is an illustration of ontology model corresponding to a third example;

[0087] FIG. 15 is an illustration of ontology model corresponding to a fourth example;

[0088] FIG. 16 is an illustration of ontology model corresponding to a fifth and sixth example;

[0089] FIG. 17 is an illustration of ontology model corresponding to a seventh example.

[0090] FIG. 18 is an illustration of ontology model corresponding to an eighth example

[0091] FIG. 19 is an illustration of ontology model corresponding to a ninth example

[0092] FIG. 20 is an illustration of ontology model corresponding to a tenth example;

[0093] FIG. 21 is an illustration of ontology model corresponding to an eleventh example;

[0094] FIG. 22 is an illustration of ontology model corresponding to a twelfth and seventeenth example.

[0095] FIG. 23 is an illustration of ontology model corresponding to a thirteenth example

[0096] FIG. 24 is an illustration of ontology model corresponding to a fourteenth example

[0097] FIG. 25 is an illustration of ontology model corresponding to a twenty-second example; and

[0098] FIG. 26 is an illustration of ontology model corresponding to a twenty-third example.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

[0099] The present invention concerns deriving transformations for transforming data conforming with one data schema to data conforming to another data schema. Preferred embodiments of the invention are described herein with respect to table-based data schema, such as RDBS and document-based schema, such as XML schema. **[0100]** Reference is now made to **FIG. 1**, which is a simplified flowchart of a method for deriving transformations for transforming data from one schema to another, in accordance with a preferred embodiment of the present invention. The flowchart begins at step **110**. At step, **120** a source data schema and a target data schema are imported. These data schema describe templates for storing data, such as templates for tables and table columns, and templates for structured documents. If necessary, the source data schema and/or the target data schema may be converted from a standard format to an internal format. For example, they may be converted from Oracle format to an internal format.

[0101] At steps 130-160 a common ontology model is obtained, into which the source data schema and the target data schema can both be embedded, At step 130 a determination is made as to whether or not an initial ontology model is to be imported. If not, logic passes directly to step 160. Otherwise, at step 140 an initial ontology model is imported. If necessary, the initial ontology model may be converted from a standard format, such as one of the formats mentioned hereinabove in the Background, to an internal format.

[0102] At step **150** a determination is made as to whether or not the initial ontology model is suitable for embedding both the source and target data schema. If so, logic passes directly to step **170**. Otherwise, at step **160** a common ontology model is built. If an initial ontology model was exported, then the common ontology is preferably build by editing the initial ontology model; specifically, by adding classes and properties thereto. Otherwise, the common ontology model is built from scratch. It may be appreciated that the common ontology model may be built automatically with or without user assistance.

[0103] At step 170 the source and target data schema are mapped into the common ontology model, and mappings therefor are generated. At step 180 a transformation is derived for transforming data conforming with the source data schema into data conforming with the target data schema, based on the mappings derived at step 170. Finally, the flowchart terminates at step 190.

[0104] Reference is now made to **FIG. 2**, which is a simplified block diagram of a system **200** for deriving transformations for transforming data from one schema to another, in accordance with a preferred embodiment of the present invention. Shown in **FIG. 2** is a schema receiver **210** for importing a source data schema and a target data schema. These data schema describe templates for storing data, such as templates for tables and table columns, and templates for structured documents. If necessary, schema receiver **210** converts the source and target data schema from an external format to an internal format.

[0105] Also shown in FIG. 2 is an ontology receiver/ builder 220 for obtaining a common ontology model, into which the source data schema and the target data schema can both be embedded. The operation of ontology receiver/ builder 220 is described hereinabove in steps 130-160 of FIG. 1.

[0106] The source and target data schema, and the common ontology model are used by a mapping processor **230** to generate respective source and target mappings, for mapping the source data schema into the common model and for mapping the target data schema into the common

ontology model. In a preferred embodiment of the present invention, mapping processor 230 includes a class identifier 240 for identifying ontology classes with corresponding to components of the source and target data schema, and a property identifier 250 for identifying ontology properties corresponding to other components of the source and target data schema, as described in detail hereinbelow.

[0107] Preferably, the source and target mappings generated by mapping processor, and the imported source and target data schema are used by a transformation generator **260** to derive a source-to-target transformation, for transforming data conforming to the source data schema into data conforming to the target data schema.

[0108] Reference is now made to FIG. 3, which is a simplified flowchart of a method for building a common ontology model into which one or more data schema can be embedded, in accordance with a preferred embodiment of the present invention. The flowchart begins are step 310. Steps 120, 140 and 160 are similar to these same steps in FIG. 1, as described hereinabove. Finally, the flowchart terminates at step 320.

[0109] Reference is now made to FIG. 4, which is a simplified block diagram of a system 400 for building a common ontology model into which one or more data schema can be embedded, in accordance with a preferred embodiment of the present invention. Shown in FIG. 4 is schema receiver 210 from FIG. 2 for importing data schema. Also shown in FIG. 4 is an ontology receiver 420, for importing an initial ontology model. If necessary, ontology receiver 420 converts the initial ontology model from an external format to an internal format.

[0110] The initial ontology model and the imported data schema are used by an ontology builder **430** for generating a common ontology model, into which the imported data schema can all be embedded. In a preferred embodiment of the present invention, ontology builder **430** generates the common ontology model by editing the initial ontology model; specifically, by using a class builder **440** to add classes thereto based on components of the imported data schema, and by using a property builder **450** to add properties thereto based on other components of the imported data schema.

[0111] Applications of the present invention include inter alia:

- **[0112]** integrating between two or more applications that need to share data;
- **[0113]** transmitting data from a database schema across a supply chain to a supplier or customer using a different database schema;
- **[0114]** moving data from two or more databases with different schemas into a common database, in order that queries may be performed across the two or more databases;
- [0115] loading a data warehouse database for off-line analysis of data from multiple databases;
- [0116] synchronizing two databases;
- [0117] migrating data when a database schema is updated;

[0118] moving data from an old database or database application to a replacement database or database application, respectively.

[0119] Relational Database Schema

[0120] Relational database schema (RDBS), also referred to as table definitions or, in some instances, metadata, are used to define templates for organizing data into tables and table columns, also referred to as fields. Often multiple schema exist for the same source of data, and as such the data cannot readily be imported or exported from one application to another. The present invention describes a general method and system for transforming an input, or source relational database schema into an output, or target schema. In a preferred embodiment, the present invention can be used to provide an SQL query, which when applied to a relational database from the source schema, produces a relational database in the target schema.

[0121] As described in detail hereinbelow, the present invention preferably uses an ontology model to determine an SQL query that accomplishes a desired source to target transformation. Specifically, the present invention employs a common ontology model into which both the source RDBS and target RDBS can be mapped. By mapping the source and target RDBS into a common ontology model, the present invention derives interrelationships to determine a suitable SQL query for transforming databases conforming with the target RDBS.

[0122] The present invention can also be used to derive executable code that transforms source relational databases into the target relational databases. In a preferred embodiment, the present invention creates a Java program that executes the SQL query using the JDBC (Java Database Connectivity) library. In an alternative embodiment the Java program manipulates the databases directly, without use of an SQL query.

[0123] For enterprise applications, SQL queries generated by the present invention are preferably deployed within an Enterprise Application Integration infrastructure.

[0124] Although the source and target RDBS are mapped into a common ontology model, the derived transformations of the present invention go directly from source RDBS to target RDBS without having to transform data via an ontological format. In distinction, prior art Universal Data Model approaches transform via a neutral model.

[0125] The present invention applies to N relational database schema, where $N \ge 2$. Using the present invention, by mapping the RDBS into a common ontology model, data can be moved from any one of the RDBS to any other one. In distinction to prior art approaches that require on the order of N^2 mappings, the present invention requires at most N mappings.

[0126] A "mapping" from an RDBS into an ontology model is defined as:

[0127] (i) an association of each table from the RDBS with a class in the ontology model, in such a way that rows of the table correspond to instances of the class; and

[0128] (ii) for each given table from the RDBS, an association of each column of the table with a property or a composition of properties in the ontology model, the source of which is the class corresponding to the given table and the target of which has a data type that is compatible with the data type of the column.

[0129] A mapping from an RDBS into an ontology model need not be surjective. That is, there may be classes and properties in the ontology that do not correspond to tables and columns, respectively, in the RDBS. A mapping is useful in providing a graph representation of an RDBS.

[0130] In general, although a mapping from an RDBS into an ontology model may exist, the nomenclature used in the RDBS may differ entirely from that used in the ontology model. Part of the utility of the mapping is being able to translate between RDBS language and ontology language. It may be appreciated by those skilled in the art, that in addition to translating between RDBS table/column language and ontology class/property language, a mapping is also useful in translating between queries from an ontology query language and queries from an RDBS language such as SQL (standard query language).

[0131] Reference is now made to **FIG. 5**, which is a first simplified illustration of a mapping from an RDBS into an ontology model, in accordance with a preferred embodiment of the present invention. Shown in **FIG. 5** is a table **500**, denoted T1, having four columns denoted C1, C2, C3 and C4. Also shown in **FIG. 1** is an ontology model **550** having a class denoted K1 and properties P1, P2, P3 and P4 defined on class T1. The labeling indicates a mapping from table T1 into class K1, and from columns C1, C2, C3 and C4 into respective properties P1, P2, P3 and P4.

[0132] Reference is now made to **FIG. 6**, which is a second simplified illustration of a mapping from an RDBS into an ontology model, in accordance with a preferred embodiment of the present invention. Shown in **FIG. 6** are table T1 from **FIG. 5**, and a second table **600**, denoted T2, having four columns denoted D1, D2, D3 and D4. Column C1 of table T1 is a key; i.e., each entry for column C1 is unique, and can be used as an identifier for the row in which it is situated. Column D3 of table T2 refers to table T1, by use of the key from column C1. That is, each entry of column D3 refers to a row within table T1, and specifies such row by use of the key from C1 for the row.

[0133] Also shown in **FIG. 6** is an ontology model **650** having two classes, denoted K1 and K2. Class K1 has properties P1, P2, P3 and P4 defined thereon, and class K2 has properties Q1, Q2, Q4 and S defined thereon. Property S has as its source class K1 and as its target class K2. The labeling indicates a mapping from table T1 into class K1, and from columns C1, C2, C3 and C4 into respective properties P1, P2, P3 and P4. The fact that C1 serves as a key corresponds to property P1 being one-to-one, so that no two distinct instances of class K1 have the same values for property P1.

[0134] The labeling also indicates a mapping from table T2 into class K2, and from columns D1, D2 and D4 into respective properties Q1, Q2 and Q4. Column D3 corresponds to a composite property P1oS, where o denotes function composition. In other words, column D3 corresponds to property P1 of S(K2).

[0135] The targets of properties P1, P2, P3, P4, Q1, Q2 and Q4 are not shown in **FIG. 6**, since these properties preferably map into fundamental types corresponding to the data types of the corresponding columns entries. For example, the target of P1 may be an integer, the target of P2 may be a floating point number, and the target of P3 may be a character string. Classes for such fundamental types are not shown in order to focus on more essential parts of ontology model **650**.

[0136] Classes K1 and K2, and property S are indicated with dotted lines in ontology model **650**. These parts of the ontology are transparent to the RDBS underlying tables T1 and T2. They represent additional structure present in the ontology model which is not directly present in the RDBS.

[0137] Given a source RDBS and a target RDBS, in a preferred embodiment of the present invention an appropriate transformation of source to target RDBS is generated by:

- **[0138]** (i) mapping the source and target RDBS into a common ontology model;
- **[0139]** (ii) representing fields of the source and target RDBS in terms of properties of the ontology model, using symbols for properties;
- **[0140]** (iii) deriving expressions for target symbols in terms of source symbols; and
- **[0141]** (iv) converting the expressions into one or more SQL queries.

[0142] Reference is now made to **FIG. 7**, which is a simplified illustration of relational database transformations involving constraints and joins, in accordance with a preferred embodiment of the present invention.

[0143] XML Schema

[0144] As described in detail hereinbelow, the present invention preferably uses an ontology model to determine an XSLT transformation that accomplishes a desired source to target transformation. Specifically, the present invention employs a common ontology model into which both the source XML schema and target XML schema can be mapped. By mapping the source and target XML schema into a common ontology model, the present invention derives interrelationships among their elements and attributes, and uses the interrelationships to determine suitable XSLT script for transforming documents generating documents conforming with the source XML schema.

[0145] The present invention can also be used to derive executable code that transforms source XML documents into the target XML documents. In a preferred embodiment, the present invention packages the derived XSLT script with a Java XSLT engine to provide an executable piece of Java code that can execute the transformation.

[0146] Preferably, this is used to deploy XSLTs within an EAI product such as Tibco. Specifically, in a preferred embodiment of the present invention, a function (similar to a plug-in) is installed in a Tibco MessageBroker, which uses the Xalan XSLT engine to run XSLT scripts that are presented in text form. As an optimization, the XSLT script files are preferably compiled to Java classfiles.

[0147] Reference is now made to **FIG. 8**, which is a simplified illustration of use of a preferred embodiment of the present invention to deploy XSLT scripts within an EAI product such as Tibco.

[0148] User Interface

[0149] Applicant has developed a software application, named COHERENCETM, which implements a preferred embodiment of the present invention to transform data from one schema to another. Coherence enables a user

- [0150] to import source and target RDBS;
- **[0151]** to build an ontology model into which both the source and target RDBS can be mapped;
- [0152] to map the source and target RDBS into the ontology model; and
- **[0153]** to impose constraints on properties of the ontology model.

[0154] Once the mappings are defined, Coherence generates an SQL query to transform the source RDBS into the target RDBS.

[0155] Reference is now made to FIGS. 9A-9E, which are illustrations of a user interface for transforming data from one relational database schema to another using the Coherence software application, in accordance with a preferred embodiment of the present invention. Shown in FIG. 9A is a main Coherence window 905 with a left pane 910 and a right pane 915. Window 905 includes three primary tabs 920, 925 and 930, labeled Authoring, Mapping and Transformations, respectively. Authoring tab 920 is invoked in order to display information about the ontology model, and to modify the model by adding, deleting and editing classes and properties. Mapping tab 925 is invoked in order to display information about the RDBS and the mappings of the RDBS into the ontology, and to edit the mappings. Transformations tab 930 is invoked to display transformations in the form of SQL queries, from a source RDBS into a target RDBS. In FIG. 9A, tab 920 for Authoring is shown selected.

[0156] Left pane 910 includes icons for two modes of viewing an ontology: icon 935 for viewing in inheritance tree display mode, and icon 940 for viewing in package display mode.

[0157] Inheritance tree display mode shows the classes of the ontology in a hierarchical fashion corresponding to superclass and subclass relationships. As illustrated in FIG. 9A, in addition to the fundamental classes for Date, Number, Ratio, String and NamedElement, there is a class for City. Corresponding to the class selected in left pane 910, right pane 915 displays information about the selected class. Right pane 915 includes six tabs for class information display: tab 945 for General, tab 950 for Properties, tab 955 for Subclasses, tab 960 for Enumerated Values, tab 965 for Relations and tab 970 for XML schema. Shown in FIG. 9A is a display under tab 945 for General. The display includes the name of the class, Being, and the package to which it belongs; namely, fundamental. Also shown in the display is a list of immediate superclasses, which is an empty list for class Being. Also shown in the display is a textual description of the class; namely, that Being is a root class for all classes.

[0158] Tab **960** for Enumerated Values applies to classes with named elements; i.e., classes that include a list of all possible instances. For example, a class Boolean has enumerated values "True" and "False," and a class Gender may have enumerated values "Male" and "Female."

[0159] FIG. 9B illustrates package display mode for the ontology. Packages are groups including one or more ontology concepts, such as classes, and properties. Packages are used to organize information about an ontology into various groupings. As illustrated in **FIG. 9B**, there is a fundamental package that includes fundamental classes, such as Being, Boolean, Date and Integer. Also shown in **FIG. 9B** is a package named WeatherFahrenheit, which includes a class named City.

[0160] As shown in FIG. 9B, City is selected in left pane 910 and, correspondingly, right pane 915 displays information about the class City. Right pane 915 display information under Tab 950 for Properties. As can be seen, class City belongs to the package WeatherFahrenheit, and has four properties; namely, Celsius of type RealNumber, city of type String, Fahrenheit of type RealNumber and year of type RealNumber. FIG. 9B indicates that the property Celsius satisfies a constraint. Specifically, Celsius=5*(Fahrenheit -32)/9.

[0161] In **FIG. 9C**, the tab **925** for Mapping is shown selected. As shown in the left pane of **FIG. 9C**, two RDBS have been imported into Coherence. A first RDBS named WeatherCelsius, which includes a table named Towns, and a second RDBS named WeatherFahrenheit, which includes a table named Cities.

[0162] The table named Cities is shown selected in **FIG. 9C**, and correspondingly the right pane display information regarding the mapping of Cities into the ontology. As can be seen, the table Cities contains three fields; namely, Fahrenheit, city and year. The table Cities has been mapped into the ontology class City, the field Fahrenheit has been mapped into the ontology property Fahrenheit, the field city has been mapped into the ontology property name, and the field year has been mapped into the ontology property year. The RDBS WeatherFahrenheit will be designated as the source RDBS.

[0163] When tab 925 for Mapping is selected, the right pane includes three tabs for displaying information about the RDBS: tab 975 for Map Info, tab 980 for Table Info and tab 985 for Foreign Keys.

[0164] The RDBS named WeatherCelsius is displayed in **FIG. 9D**. As can be seen, the table Towns contains three fields; namely, town, Celcius and year. The table Towns has been mapped into the ontology class City, the field town has been mapped into the ontology property name, the field Celcius has been mapped into the ontology property Celcius, and the field year had been mapped into the ontology property Celcius will be designated as the target RDBS.

[0165] As such, the target RDBS is

TABLE I

	Towns		
Town	Celcius	Year	

[0166] and the source RDBS is

TABLE II

	Cities	
Fahrenheit	City	Year

[0167] In **FIG. 9E**, the tab **930** for Transformations is shown selected. As can be seen in the right pane, the source table is Cities and the target table is Towns. The SQL query

INSERT INTO (SELECT	WeatherCelcius.Towns(CELCIUS, TOWN, YEAR)
`	(5 * (A.FAHRENHEIT – 32)/9) AS CELCIUS, A.CITY AS TOWN, A.YEAR AS YEAR
FROM	WeatherFahrenheit.Cities A);

[0168] accomplishes the desired transformation.

[0169] Reference is now made to **FIG. 10**, which is an illustration of a user interface for an application that imports an RDBS into Coherence, in accordance with a preferred embodiment of the present invention. Shown in **FIG. 10** is a window **1010** for a schema convertor application. Preferably, a user specifies the following fields:

- [0170] Database Name 1020: What Oracle refers to as an SID (System Identifier).
- [0171] Host Name 1030: The name of an Oracle 8i server (or Global Database Name).
- [0172] Port 1040: Port number
- **[0173]** Username **1050**: The username of a user with privileges to the relevant schemas.
- **[0174]** Password **1060**: The password of the user with privileges to the relevant schemas.
- [0175] Oracle schema 1070: The schema or database in Oracle to be converted to .SML format. The .SML format is an internal RDBS format used by Coherence. When importing more than one schema, a semicolon (;) is placed between schema names.
- **[0176]** Coherence schema **2080**: The label identifying the RDBS that is displayed on the Mapping Tab in Coherence. This field is optional; if left blank, the Oracle schema name will be used.
- [0177] Output File 1090: A name for the .SML file generated.

[0178] Reference is now made to **FIGS. 11A-11R**, which are illustrations of a for transforming data from one XML schema to another using the Coherence software application, in accordance with a preferred embodiment of the present invention. Shown in **FIG. 11A** is a window with package view of an Airline Integration ontology model in its left lane. The left pane displays classes from a fundamental package. A class Date is shown highlighted, and its properties are shown in the right pane. Fundamental packages are used for standard data types. Shown in **FIG. 11B** is a window with a hierarchical view of the Airline Integration ontology model

9

in its left pane. The left pane indicates that FrequentFlyer is a subclass of Passenger, Passenger is a subclass of Person, and Person is a subclass of Being. The right pane displays general information about the class FrequentFlyer.

[0179] FIG. 11C shows a window used for opening an existing ontology model. In the Coherence software application, ontology models are described using XML and stored in .oml files. Such files are described in applicant's co-pending patent application U.S. Ser. No. 09/866,101 filed on May 25, 2001 and entitled METHOD AND SYSTEM FOR COLLABORATIVE ONTOLOGY MODELING, the contents of which are hereby incorporated by reference.

[0180] FIG. 11D shows the hierarchical view from FIG. 11B, indicating properties of the FrequentFlyer class. The property fullName is highlighted, and a window for constraint information indicates that there is a relationship among the ontology properties firstName, lastName and fullName; namely, that fullName is the concatenation of firstName and lastName with a white space therebetween. This relationship is denoted as Constraint_5.

[0181] FIG. 11E shows the hierarchical view from **FIG. 11B**, indicating test instance of the Passenger class. A list of instances is displayed in the right pane, along with property values for a specific selected instance from the list.

[0182] FIG. 11F shows two imported XML schema for airline information. FIG. 11G shows a window for importing XML schema into Coherence. FIG. 11H shows a window with a display of an imported XML schema for British Airways, with a list of complexTypes from the imported schema. The complexType Journey is selected, and the right pane indicates that Journey and its elements are currently not mapped to a class and properties of the ontology model.

[0183] FIG. 11I shows a window for generating a mapping from the British Airways XML schema into the Airline Integration ontology model. The ontology class Flight is shown selected to correspond to the XML ComplexType Journey. FIG. 11J shows the left pane from FIG. 11H, with the right pane now indicating that the XML complexType Journey from the British Airways XML schema has been mapped to the class Flight from the Airline Integration ontology model. FIG. 11K shows the left pane from FIG. 11H, with a window for selecting properties and indirect properties (i.e., compositions of properties) to correspond to elements from the XML schema. Shown selected in FIG. 11K is a property distanceInMiles() of the class Flight. FIG. 11L shows the left pane from FIG. 11H, with the right pane now indicated that Journey has been mapped to Flight, and the XML element distance in miles within the complex-Type Journey has been mapped to the property distanceIn-Miles() of the class Flight. FIG. 11M shows the left pane from FIG. 11H, with the right pane now indicating that the mapping has been extended to all XML elements of the complexType Journey, showing the respective properties to which each element is mapped. FIG. 11N shows schema info for the complexType Journey, listing its elements and their data types.

[0184] FIG. 11O shows a window for specifying a transformation to be derived. Shown in **FIG. 10** is a request to derive a transformation from a source data schema, namely, the imported SwissAir XML schema to a target data schema,

namely, the imported British Airways XML schema. Shown in **FIG. 11P** is an XSLT script generated to transform XML documents conforming to the SwissAir schema to XML documents conforming to the British Airways schema. **FIG. 11Q** shows a specific transformation of a SwissAir XML document to a British Airways XML document, obtained by applying the derived XSLT script from **FIG. 11P**. Finally, **FIG. 11R** shows a display of the newly generated British Airways XML document with specific flights and passengers.

EXAMPLES

[0185] For purposes of clarity and exposition, the workings of the present invention are described first through a series of twenty-three examples, followed by a general description of implementation. Two series of examples are presented. The first series, comprising the first eleven examples, relates to RDBS transformations. For each of these examples, a source RDBS and target RDBS are presented as input, along with mappings of these schema into a common ontology model. The output is an appropriate SQL query that transforms database tables that conform to the source RDBS, into database tables that conform to the target RDBS. Each example steps through derivation of source and target symbols, expression of target symbols in terms of source symbols and derivation of an appropriate SQL query based on the expressions.

[0186] The second series of examples, comprising the last twelve examples, relates to XSLT transformation. For each of these examples, a source XML schema and target XML schema are presented as input, along with mappings of these schema into a common ontology model. The output is an appropriate XSLT script that transforms XML documents that conform to the source schema into XML documents that conform to the target schema.

A First Example

Schoolchildren

[0187] In a first example, a target table is of the following form:

TABLE III

	Target Table T for	First Example	
Child_Name	Mother_Name	School_Location	Form

[0188] Four source tables are given as follows:

TABLE IV

	Source Table S ₁ for Fi	rst Example
Name	School_Attending	Mother_NI_Number

[0189]

TABLE V

Source Table S_2 for First Example			
NI_Number	Name	Region	Car_Number
[0190]			
	TAB	LE VI	
Se	ource Table S ₃	for First Exa	ample
Name	Locatio	n	HeadTeacher
[0191]			
	TAB	LE VII	
Se	ource Table S ₄	for First Exa	ample
Name	Ye	ar	Form

[0192] The underlying ontology is illustrated in **FIG. 12**. The dotted portions of the ontology in **FIG. 12** show additional ontology structure that is transparent to the relational database schema. Using the numbering of properties

indicated in FIG. 12, the unique properties of the ontology are identified as:

IADLE VIII	E VIII	Æ	ГАВІ
------------	--------	---	------

Unique Properties within Ontology for First Example	
Property	Property Index
name(Child) national_insurance_number(Person) name(School)	6 4 10

[0193] The mapping of the target schema into the ontology is as follows:

TABLE IX

Mapping from Target schema to Ontology for First Example Property Index schema Ontology Т Class: Child T.Child_Name Property: name(Child) 6 T.Mother_Name Property: name(mother(Child)) 305 T.School_Location Property: 1209 location(school_attending(Child)) T.Form Property: current_school_form(Child) 8

[0194] The symbol o is used to indicate composition of properties. The mapping of the source schema into the ontology is as follows:

TABLE X

Mapping from Source schema to Ontology for First Example Property

schema	Ontology	Index
S ₁	Class: Child	
S ₁ .Name	Property: name(Child)	6
S1.School_Attending	Property: name(school_attending(Child))	1009
S1.Mother_NI_Number	Property: national_insurance_number(mother(Child))	405
S_2	Class: Person	
S2.NI_Number	Property: national_insurance_number(Person)	4
S ₂ .Name	Property: name(Person)	3
S ₂ .Region	Property: region_of_residence(Person)	1
S2.Car_Number	Property: car_registration_number(Person)	2
S ₃	Class: School	
S ₃ .Name	Property: name(School)	10
S ₃ .Location	Property: location(School)	12
S ₃ .HeadTeacher	Property: name(headteacher(School))	3011
S_4	Class: Child	
S ₄ .Name	Property: name(Child)	6
S ₄ .Year	Property: year_of_schooling(Child}	7
S ₄ .Form	Property: current_school_form(Child)	8

Source Symb	ools for First Example
Source Table	Source Symbols
S ₁	100906^{-1} 40506^{-1}
S_2	304^{-1} 104^{-1} 204^{-1}
S ₃	12010 ⁻¹ 3011010 ⁻¹
S_4	706 ⁻¹ 806 ⁻¹

TABLE XI

[0196] The symbols in Table XI relate fields of a source table to a key field. Thus in table S_1 the first field, S_1 .Name is a key field. The second field, S_1 .School_Attending is related to the first field by the composition 100906^{-1} , and the third field, S_1 .Mother_NI_Number is related to the first field by the composition 40506^{-1} . In general, if a table contains more than one key field, then expressions relative to each of the key fields are listed.

[0197] The inverse notation, such as 6^{-1} is used to indicate the inverse of property 6. This is well defined since property 6 is a unique, or one-to-one, property in the ontology model. The indices of the target properties, keyed on Child_Name are:

TABLE XII

Target Symbols for First Example		
Target Table	Target Symbols	Paths
Т	30506 ⁻¹ 120906 ⁻¹ 806 ⁻¹	$(304^{-1}) \circ (40506^{-1})$ $(12010^{-1}) \circ (100906^{-1})$ (806^{-1})

[0198] Based on the paths given in Table XII, the desired SQL query is:

INSERT INTO TO (SELECT	(Child_Name, Mother_Name, School_Location, Form)
	S ₁ .Name AS Child_Name,
	S ₂ .Name AS Mother_Name,
	S ₃ .Location AS School_Location,
	S ₄ .Form AS Form
FROM	•
	S ₁ , S ₂ , S ₃ , S ₄
WHERE	1. 2. 0. 1
	S ₂ .NI_Number = S ₁ .Mother_NI_Number AND
	S_3 .Name = S_1 .School_Attending AND
	S_4 .Name = S_1 .Name);

[0199] The rules provided with the examples relate to the stage of converting expressions of target symbols in terms of source symbols, into SQL queries. In general,

[0200] Rule 1: When a target symbol is represented using a source symbols, say (aob^{-1}) from a source table, S, then the column of S mapping to a is used in the SELECT clause of the SOL query and the column of S mapping to b is used in the WHERE clause.

[0201] Rule 2: When a target symbol is represented as a composition of source symbols, say $(aob^{-1}) \circ (boc^{-1})$, where aob^{-1} is taken from a first source table, say S_1 , and boc^{-1} is taken from a second source table, say S_2 , then S_1 and S_2 must be joined in the SQL query by the respective columns mapping to b.

[0202] Rule 3: When a target symbol is represented using a source symbols, say (aob^{-1}) , from a source table, S, and is not composed with another source symbol of the form boc^{-1} , then table S must be joined to the target table through the column mapping to b.

[0203] When applied to the following sample source data, Tables XIII, XIV, XV and XVI, the above SQL query produces the target data in Table XVII.

TABLE XIII

Sample Source Table S ₁ for First Example			
Name	School_Attending	Mother_NI_Number	
Daniel Ashton	Chelsea Secondary School	123456	
Peter Brown	Warwick School for Boys	673986	
Ian Butler	Warwick School for Boys	234978	
Matthew Davies	Manchester Grammar School	853076	
Alex Douglas	Weatfields Secondary School	862085	
Emma Harrison	Camden School for Girls	275398	
Martina Howard	Camden School for Girls	456398	

TABLE XIV

Sample Source Table S2 for First Example				
NI_Number	Name	Region	Car_Number	
123456	Linda	London	NULL	
673986	Amanda	Warwick	NULL	
456398	Claire	Cambridgeshire	NULL	
862085	Margaret	NULL	NULL	
234978	Amanda	NULL	NULL	
853076	Victoria	Manchester	NULL	
275398	Elizabeth	London	NULL	

[0205]

TABLE XV

Sample Source Table S ₃ for First Example				
Name		Location	HeadTeacher	
Manchest Camden S Weatfield Chelsea S Warwick	er Grammar School School for Girls s Secondary School Secondary School School for Boys	Manchester London Cambridgeshire London Warwickshire	M. Payne J. Smith NULL I. Heath NULL	

[0206]

TABLE XVI

Sample Source Table S ₄ for First Example				
Name	Year	Form		
Peter Brown	7	Lower Fourth		
Daniel Ashton	10	Mid Fifth		
Matthew Davies	4	Lower Two		
Emma Harrison	6	Three		
James Kelly	3	One		
Greg McCarthy	5	Upper Two		
Tina Reynolds	8	Upper Fourth		

[0207]

TABLE XVII

Sample Target Table T for First Example				
Child_Name	Mother_Name	School_Location	Form	
Daniel Ashton Peter Brown Matthew Davies Emma Harrison	Linda Amanda Victoria Elizabeth	London Warwickshire Manchester London	Mid Fifth Lower Fourth Lower Two Three	

A Second Example

Employees

[0208] In a second example, a target table is of the following form:

TABI	F	XV	ЛП
IADL	ıL.	Λ	V 111

	Target Table T f	or Second Example	
Name	Department	Supervisor	Room#

[0209] Four source tables are given as follows:

TARI F	XIX
IADLE	ΛΙΛ

Source Table S ₁ for Second Example			
Emp_ID#	Name	Department	

[0210]

TABLE XX

Source Table S_2 for Second Example			
Employee_	_Name	Supervisor	Project

[0211]

TABLE XXI

Source Table	e S ₃ for Second Example	
--------------	-------------------------------------	--

ID# Room_Assignment Telephone#

[0212]

TABLE XXII

Source Table S ₄ for Second	Example
Department	Budget

[0213] The underlying ontology is illustrated in **FIG. 13**. The dotted portions of the ontology in **FIG. 13** are additional ontology structure that is transparent to the relational database schema. The unique properties of the ontology are:

TABLE XXIII

Unique Properties within Ontology for Second Example		
Property	Property Index	
name(Employee)	3	
ID#(Employee)	4	

[0214] The mapping of the target schema into the ontology is as follows:

	TA	ΒL	E	XXI	V
--	----	----	---	-----	---

Mapping from Target schema to Ontology for Second Example

schema	Ontology	Property Index
Т	Class: Employee	
T.Name	Property: name(Employee)	3
T.Department	Property: code(departmental_affiliation(Employee))	807
T.Supervisor T.Room#	Property: name(supervisor(Employee)) Property: room_number(Employee)	306 1

[0215] The mapping of the source schema into the ontology is as follows:

TABLE XXV

Mapping from Source schema to Ontology for Second Example			
schema	Ontology	Property Index	
S ₁	Class: Employee		
S ₁ .Emp_ID#	Property: ID#(Employee)	4	
S ₁ .Name	Property: name(Employee)	3	
S1.Department	Properly: code(departmental_affiliation(Employee))	807	
S_2	Class: Employee		
S2.Employee_Name	Property: name(Employee)	3	
S2.Supervisor	Property: name(supervisor(Employee))	306	
S2.Project	Property: project_assignment(Employee)	5	
S ₃	Class: Employee		
S ₃ .ID#	Property: ID#(Employee)	4	
S3.Room_Assignment	Property: room_number(Employee)	1	
S3.Telephone#	Property: tel#(Employee)	2	
S_4	Class: Department		
S ₄ .Department	Property: code(Department)	8	
S ₄ .Budget	Property: budget_amount(Department)	9	

[0216] The indices of the source properties are:

TABLE XXVI

Source Symbols for Second Example			
Source Table	Source Symbols		
S ₁	304^{-1} 80704 ⁻¹ 403 ⁻¹ 80703 ⁻¹		
S_2	30603 ⁻¹ 503 ⁻¹		
S_3	104 ⁻¹ 204 ⁻¹		
S_4	908 ⁻¹		

[0217] The indices of the target properties, keyed on Name are:

TABLE XXVII

Target Symbols for Second Example			
Target Table	Target Symbols	Paths	
Т	80703 ⁻¹ 30603 ⁻¹ 103 ⁻¹	$\begin{array}{c} (80703^{-1}) \\ (30603^{-1}) \\ (104^{-1}) \ 0 \ (403^{-1}) \end{array}$	

[0218] Based on the paths given in Table XXVII, the desired SQL query is:

INSERT INTO T(Name, Department, Supervisor, Room#) (SELECT S₁.Name AS Name, S₁.Department AS Department,

S2.Supervisor AS Supervisor,

S₃.Room_Assignment AS Room#

 S_1, S_2, S_3

FROM

NALEDE	
WHERE	
	S_2 .Employee Name = S_1 .Name AND S_3 .ID# =
	S ₁ .Emp_ID#);

[0219] It is noted that Table S_4 not required in the SQL. When applied to the following sample source data, Tables XXVIIII, XXIX and XXX, the above SQL query produces the target data in Table XXXI.

TABLE XXVIII

Sample Source Table S ₁ for Second Example			
Emp_ID#	Name	Department	
198	Patricia	SW	
247	Eric	QA	
386	Paul	IT	

[0220]

TABLE XXIX

Sample Source Table S2 for Second Example			
Employee_Name	Supervisor	Project	
Eric	John	Release 1.1	
Patricia	George	Release 1.1	
Paul	Richard	Release 1.1	

-continued

[0221]

TABLE XXX

Sample Source Table S ₃ for Second Example			
ID#	Room_Assignment	Telephone#	
386	10	106	
198	8	117	
247	7	123	

[0222]

TABLE XXXI

	Sample Target Table	-	
Name	Department	Supervisor	Room#
Patricia	SW	George	8
Eric	QA	John	7
Paul	IT	Richard	10

A Third Example

Airline Flights

[0223] In a third example, a target table is of the following form:

TABLE XXXII					
Target Table T for Third Example					
FlightID	DepartingCity	ArrivingCity			

[0224] Two source tables are given as follows:

	TAB	LE	XXXII
--	-----	----	-------

	Source Table S_1 for Thin	d Example	
Index	APName	Location	

[0225]

TABLE XXXIV					
Source Table S_2 for Third Example					
FlightID	FromAirport	ToAirport			

[0226] The underlying ontology is illustrated in FIG. 14. The dotted portions of the ontology in FIG. 14 are additional ontology structure that is transparent to the relational database schema. The unique properties of the ontology are:

TABLE XXXV

Unique Properties within C	Intology for Third Example
Property	Property Index
name(Airport)	1
ID(Flight)	6

[0227] The mapping of the target schema into the ontology is as follows:

TABLE XXXVI

Mapping from Target schema to Ontology for Third Example					
schema	Ontology	Property Index			
Т	Class: Flight				
T.FlightID	Property: ID#(Flight)	6			
T.DepartingCity	Property: location(from_airport(Flight))	204			
T.ArrivingCity	Property: location(to_airport(Flight))	205			

[0228] The mapping of the source schema into the ontology is as follows:

TABLE XXXVII

Mapping from Source schema to Ontology for Third Example			
schema	Ontology	Property Index	
S ₁	Class: Airport		
S ₁ .Index	Property: Index(Airport)	3	
S ₁ .APName	Property: name(Airport)	1	
S ₁ .Location	Property: location(Airport)	2	
S ₂	Class: Flight		
S ₂ .FlightID	Property: ID#(Flight)	6	
S ₂ .FromAirport	Property: name(from_airport(Flight))	104	
S ₂ .ToAirport	Property: name(to_airport(Flight))	105	

[0229] The indices of the source properties are:

TABLE XXXVIII

Source Sy	mbols for Third Example	
Table	Source Symbols	
S ₁ S ₂	$103^{-1} \\ 203^{-1} \\ 301^{-1} \\ 201^{-1} \\ 10406^{-1}$	
	10506^{-1}	

[0230]	The	indices	of	the	target	properties,	keyed	on
FlightID	are:							

TABLE XXXIX				
Target Symbols for Third Example				
Table	Target Symbols	Paths		
Т	20406^{-1} 20506^{-1}	(201^{-1}) o (10406^{-1}) (201^{-1}) o (10506^{-1})		

[0231] Since the path (201^{-1}) appears in two rows of Table XXXIX, it is necessary to create two tables for S₁ in the SQL query. Based on the paths given in Table XXXVII, the desired SQL query is:

INSERT INTO T	(FlightID, DepartingCity, ArrivingCity)
`	S ₂ .FlightID AS FlightID,
	S ₁₁ Location AS DepartingCity,
	S ₁₂ .Location AS ArrivingCity
FROM	
	S ₁ S ₁₁ , S ₁ S ₁₂ , S ₂
WHERE	
	S_{11} .APName = S_2 .FromAirport AND S_{12} .APName = S_2 .ToAirport);

[0232] In general,

[0233] Rule 4: When the same source symbol is used multiple times in representing target symbols, each occurrence of the source symbol must refer to a different copy of the source table containing it.

[0234] When applied to the following sample source data, Tables XL and XLI, the above SQL query produces the target data in Table XLII.

TABLE XL

	ample Source Table S ₁ for	Third Example
Index	APName	Location
1	Orly	Paris
2	JFK	New York
3	LAX	Los Angeles
4	HNK	Hong Kong
5	TLV	Tel Aviv
6	Logan	Boston

[0235]

TABLE XLI

Sample Source Table S ₂ for Third Example			
FlightID	FromAirport	ToAirport	
001 002 003 004	Orly JFK TLV Logan	JFK LAX HNK TLV	

[0236]

TABLE XLII

Sample Target Table T for Third Example		
FlightID	DepartingCity	ArrivingCity
001	Paris	New York
002	New York	Los Angeles
003	Tel Aviv	Hong Kong
004	Boston	Tel Aviv

A Fourth Example

Lineage

[0237] In a fourth example, a target table is of the following form:

TABLE XLIII

	Target Table T for	Fourth Example	
ID	Name	Father_Name	

[0238] One source table is given as follows:

TABLE XLIV

Source Ta	ble S for Fourth and	l Fifth Examples	
ID	Name	Father_ID	

[0239] The underlying ontology is illustrated in **FIG. 15**. The dotted portions of the ontology in **FIG. 15** are additional ontology structure that is transparent to the relational database schema. The unique properties of the ontology are:

 TABLE XLV

 Unique Properties within Ontology for Fourth and Fifth Examples

 Property
 Property Index

 name(Person)
 1

 ID#(Person)
 2

[0240] The mapping of the target schema into the ontology is as follows:

TABLE XLVI

Mapping from Ta	arget schema to Ontology for Fourth	Example
schema	Ontology	Property Index
Г Г.ID Г.Name Г.Father_Name	Class: Person Property: ID#(Person) Property: name(Person) Property: name(father(Person))	2 1 1o3

[0241] The mapping of the source schema into the ontology is as follows:

TABLE XLVII

Ma	pping from Source schema to Onto for Fourth and Fifth Examples	logy
schema	Ontology	Property Index
5	Class: Person	
S.ID	Property: ID#(Person)	2
S.Name	Property: name(Person)	1
S.Father_ID	Property: ID#(father(Person))	203

[0242] The indices of the source properties are:

TABLE XLVIII		
Source Symbols for Fourth and Fifth Examples		
	Table	Source Symbols
	S_1	$\frac{102^{-1}}{20302^{-1}}$

[0243] The indices of the target properties, keyed on ID are:

TABLE XLIX

	Target Symbols for Fe	ourth Example
Table	Target Symbols	Paths
Т	102^{-1} 10302^{-1}	(102^{-1}) (102^{-1}) o (20302^{-1})

[0244] Based on the paths given in Table XLIX, the desired SQL query is:

INSERT INTO T(II (SELECT	INSERT INTO T(ID, Name, Father_ID) (SELECT	
,	S ₁ .ID AS ID,	
	S ₁ .Name AS Name,	
	S ₂ .ID AS Father_ID	
FROM	-	
	S S ₁ , S S ₂	
WHERE		
	S_2 .ID = S_1 .Father_ID);	

A Fifth Example

Lineage

[0245] In a fifth example, the target property of Father-_Name in the fourth example is changed to Grandfather-Name, and the target table is thus of the following form:

TABLE L

Target Table T for Fifth Example			
ID	Name	Grandfather_Name	

[0246] One source table is given as above in Table XLIV.

[0247] The underlying ontology is again illustrated in FIG. 15. The unique properties of the ontology are as above in Table XLV.

[0248] The mapping of the target schema into the ontology is as follows:

TABLE LI

Mapping from Target schema to Ontology for Fifth Example		h Example
schema	Ontology	Property Index
Т	Class: Person	
T.ID	Property: ID#(Person)	2
T.Name	Property: name(Person)	1
T.Grandfather_Name	Property: name(father(father(Person)))	10303

[0249] The mapping of the source schema into the ontology is given in Table XLVII above.

[0250] The indices of the source properties are given in Table XLVIII above.

[0251] The indices of the target properties, keyed on ID are:

TABLE LII

	Target Symbols for	r Fifth Example
Table	Target Symbols	Paths
Т	102 ⁻¹ 1030302 ⁻¹	(102^{-1}) (102^{-1}) o (20302^{-1}) o (20302^{-1})

[0252] Based on the paths given in Table LII, the desired SQL query is:

INSERT INTO T(ID,	Name, Grandfather_ID)
(SELECT	S ₁ .ID AS ID, S ₁ .Name AS Name,
FROM	
WHERE	$S S_1, S S_2, S S_3$
	$S_3.ID = S_2.Father_ID AND$ $S_2.ID = S_1.Father_ID);$

A Sixth Example

Dog Owners

[0253] In a sixth example, a target table is of the following form:

TABLE LIII

Target Table T for Sixth Example		
ID	Name	Dogs_Previous_Owner

		TABLE LIV		
		Source Table S ₁ for Sixth H	Example	
	D	Name	Dog	
[0255]		TABLE LV		

	Source Table S_2 for	Sixth Example	
Owner	Name	Previous_Owner	

[0256] The underlying ontology is illustrated in **FIG. 16**. The dotted portions of the ontology in **FIG. 16** are additional ontology structure that is transparent to the relational database schema. The unique properties of the ontology are:

INDLE LVI	ГA	BI	E	\mathbf{L}	VI
-----------	----	----	---	--------------	----

Unique Properties within	Ontology for Sixth Example
Property	Property Index
ID#(Person) name(Dog)	2 6

[0257] The mapping of the target schema into the ontology is as follows:

	TABLE LVII	
Mapping from Tar	get schema to Ontology for Sixth	Example
schema	Ontology	Property Index
Т	Class: Person	
T.ID	Property: ID#(Person)	2
T.Name	Property: name(Person)	1
T.Dogs_Previous_Owner	r Property:	503

TABLE LVII

[0258] The mapping of the source schema into the ontology is as follows:

previous_owner(dog(Person))

TABLE	LVIII
IT ID DD	T TTT

Mapping from Source schema to Ontology for Sixth Example		
schema	Ontology	Property Index
S ₁	Class: Person	
S ₁ .ID	Property: ID#(Person)	2
S ₁ .Name	Property: name(Person)	1
S ₁ .Dog	Property: name(dog(Person))	603
S_2	Class: Dog	
S ₂ .Owner	Property: name(owner(Dog))	104
S ₂ .Name	Property: name(Dog)	6
S ₂ .Previous_Owner	Property: name(previous_owner(Dog))	105

[0259] The indices of the source properties are:

TABLE LIX

Source Sy	nbols for Sixth Example
Table	Source Symbols
S ₁	102 ⁻¹ 60302 ⁻¹
S_2	10406 ⁻¹ 10506 ⁻¹

[0260] The indices of the target properties, keyed on ID are:

TABLE LX

	Target Symbols for	Sixth Example
Table	Target Symbols	Paths
Т	102 ⁻¹ 50302 ⁻¹	(102^{-1}) $(10506^{-1}) \circ (60302^{-1})$

[0261] Based on the paths given in Table LX, the desired SQL query is:

INSERT IN (SELECT	O T(ID, Name, Dogs_Previous_Owner)
	S ₁ .ID AS ID, S ₁ .Name AS Name,
	S ₂ Previous_Owner AS Dogs_Previous_Owner
FROM	- •
	S ₁ , S ₂
WHERE	
	S_2 .Name = S_1 .Dog);

A Seventh Example

Employees

[0262] In a seventh example, a target table is of the following form:

ΓAI	3L	E	L	XI
-----	----	---	---	----

Target Table T for Seventh Example			Example	
ID	Name	Email	Department	

[0263] Five source tables are given as follows:

_

TABI	E.	LX	H
	_		

Source Tal	ble S ₁ for Seventh Example
ID	Department

ID

ID

[0264]

TABLE LXIII

Source 7	Fable :	S_2	for	Seventh	Example
----------	---------	-------	-----	---------	---------

Email

Name

[0265]

TA	BLE	E LX	ΊV	

Source Table ${\rm S}_3$ for Seventh Example

[0266]

TABLE LXV

Source Table S ₄ for Sev	venth Example
ID	Email

[0267]

T	ABLE LXVI	
Source Table S_5 for Seventh Example		
ID	Department	

[0268] The underlying ontology is illustrated in **FIG. 17**. The dotted portions of the ontology in **FIG. 17** are additional ontology structure that is transparent to the relational database schema. The unique properties of the ontology are:

TABLE LXVII Unique Properties within Ontology for Seventh Example Property Property Index ID#(Person) 2

[0269] The mapping of the target schema into the ontology is as follows:

TABLE LXVIII

Mapping from Target schema to Ontology for Seventh Example

schema	Ontology	Property Index
Т	Class: Person	
T.ID	Property: ID#(Person)	2
T.Name	Property: name(Person)	1
T.Email	Property: e-mail(Person)	3
T.Department	Property: department(Person)	4

[0270] The mapping of the source schema into the ontology is as follows:

TABLE LXIX

schema	Ontology	Property Index
S ₁	Class: Employee	
S ₁ .ID	Property: ID#(Employee)	2
S ₁ .Department	Property: department(Employee)	4
S ₂	Class: Employee	
S ₂ .ID	Properly: ID#(Employee)	2
S ₂ .Email	Property: e-mail(Employee)	3
S ₃	Class: Employee	
S ₃ .ID	Property: ID#(Employee)	2
S ₃ .Name	Property: name(Employee)	1
S4	Class: Employee	
S ₄ .ID	Property: ID#(Employee)	2
S ₄ .Email	Property: e-mail(Employee)	3
S ₅	Class: Employee	
S ₅ .ID	Property: ID#(Employee)	2
S ₅ .Department	Property: department(Employee)	4

[0271] The indices of the source properties are:

TABLE LXX

Source Syml	ools for Seventh Example
Table	Source Symbols
S ₁	4o2 ⁻¹
S2	302-1
S3	102^{-1}
5 ₄	302-1
3 ₅	4o2 ⁻¹

[0272] The indices of the target properties, keyed on ID are:

TABLE LXXI		
Target Symbols for Seventh Example		
Table	Target Symbols	Paths
Т	$102^{-1}302^{-1}402^{-1}$	$(102^{-1}) (302^{-1}) (402^{-1})$

[0273] Based on the paths given in Table LXXI, the desired SQL query is:

INSERT IN (SELECT	TO T(ID, Name, Email, Department)
	S ₁ .ID AS ID, S ₃ .Name AS Name,
	S ₂ .Email AS Email,
	S ₁ .Department AS Department
FROM	
	S ₁ , S ₂ , S ₃

[0278]

-continued			
WHERE			
	S_2 .ID = S_1 .ID AND S_3 .ID = S_1 .ID		
UNION			
SELECT			
	S_1 .ID AS ID,		
	S ₃ .Name AS Name,		
	S ₄ .Email AS Email,		
FROM	S ₁ .Department AS Department		
FROM			
WHEDE	S_1, S_3, S_4		
WHEKE			
UNION	$S_3.ID = S_1.ID \text{ AND } S_4.ID = S_1.ID$		
CELECT			
SELECT	S ID AS ID		
	S ₁ .ID AS ID,		
	S ₃ .Ivalite AS Ivalite,		
	S Department AS Department		
FROM	55.Department AS Department		
IROM	2 2 2 2		
WHERE	51, 52, 53, 55		
(TIDILE)	S_2 , ID = S_4 , ID and S_2 , ID = S_4 , ID and S_5 , ID = S_4 , ID		
UNION	-213131		
SELECT			
	S ₁ .ID AS ID,		
	S ₃ .Name AS Name,		
	S ₄ .Email AS Email,		
	S ₅ .Department AS Department		
FROM			
	S ₁ , S ₃ , S ₄ , S ₅		
WHERE			
	S_2 .ID = S_1 .ID and S_3 .ID = S_1 .ID and		
	S_4 .ID = S_1 .ID AND S_5 .ID = S_1 .ID);		

02/4 In general,	[0274]	In general.
------------------	--------	-------------

[0275] Rule 5: When a source symbol used to represent a target symbol is present in multiple source tables each such table must be referenced in an SOL querv and the resultant queries joined.

[0276] When applied to the following sample source data, Tables LXXII, LXXIII, LXXIV, LXXV and LXXVI, the above SQL query produces the target data in Table LXXVII.

TABLE LXXII		
Sample Source Table S ₁ for Seventh Example		
ID	Department	
123 456 789	SW PdM SW	

[0277]

TABLE LXXIII

Sample Source Table S ₂ for Seventh Example		
ID	Email	
123 456 789	jack@company jan@company jill@company	

TABLE LXXIV		
Sample Source Table S ₃ for Seventh Example		
ID	Name	
123	Jack	
456	Jan	
789	Jill	
999	Joe	
111	Jim	
888	Jeffrey	

[0279]

TABLE LXXV

Sample Source Tal	ble S ₄ for Seventh Example
ID	Email
999 111 888	joe@ company jim@ company Jeffrey@ company

[0280]

TABLE LXXVI

Sample Source Ta	ible S ₅ for Seventh Example
ID	Department
999 111 888	Sales Business_Dev PdM

[0281]

TABLE LXXVII

ID	<u>Sample T</u>	arget Table T for Seventh	<u>Example</u>
	Name	Email	Department
123	Jack	jack@company	SW
456	Jan	jan@company	PdM
789	Jill	jill@company	SW
111	Jim	jim@company	Business_Dev
888	Jeffrey	jeffrey@company	PdM
999	Joe	joe@company	Sales

An Eighth Example

Employees

[0282] In an eighth example, a target table is of the following form:

TABLE LXXVIII

	Target Table T for Eighth	Example
Emp_Name	Emp_Division	Emp_Tel_No

[0283]	Two source	tables	are given	as follows:
--------	------------	--------	-----------	-------------

	TABLE LX	XIX	
So	urce Table S ₁ for Ei	ghth Example	
Employee_Division	Employee_Tel#	Employee_Name	Room#
F03841			

[0284]

	TABLE LXXX	
Source Table S_2 for Eighth Example		
Name	Employee_Tel	Division

[0285] The underlying ontology is illustrated in **FIG. 18**. The dotted portions of the ontology in **FIG. 18** are additional ontology structure that is transparent to the relational database schema. The unique properties of the ontology are:

	TA	BL	E	L	XX	XX	
--	----	----	---	---	----	----	--

Unique Properties within Or	ntology for Eighth Example
Property	Property Index
name(Employee)	1

[0286] The mapping of the target schema into the ontology is as follows:

TABLE LXXXII

<u>Mapping from Ta</u>	Ontology	h Example Property Index
Т	Class: Employee	
T.Emp_Name	Property: name(Employee)	1
T.Emp_Division	Property: division (Employee)	4
T.Emp_Tel_No	Property: telephone_number (Employee)	2

[0287] The mapping of the source schema into the ontology is as follows:

TABLE LXXXIII

Mapping from Source schema to Ontology for Eighth Example			
schema	Ontology	Property Index	
S ₁	Class: Employee		
S1.Employee_Division	Property: division	4	
	(Employee)		
S1.Employee_Tel#	Property: telephone_number	2	
	(Employee)		
S ₁ .Employee_Name	Property: name(Employee)	1	
S1.Employee_Room#	Property: room_number	3	
	(Employee)		
S ₂	Class: Employee		
S ₂ .Name	Property: name(Employee)	1	

TABLE LXXXIII-continued

Mapping from Sour	Example	
schema	Ontology	Property Index
S ₂ .Employee_Tel	Property: telephone_number (Employee)	2
S_2 .Division	Property: division(Employee)	4

[0288] The indices of the source properties are:

TABLE LXXXIV

Source Symbols for Eighth Example		
Table	Source Symbols	
S ₁ S ₂	$ \begin{array}{c} 401^{-1} \\ 201^{-1} \\ 301^{-1} \\ 201^{-1} \\ 401^{-1} \end{array} $	

[0289] The indices of the target properties, keyed on Emp_Name are:

TABLE LXXXV

Tar	get Symbols for Eighth Ex	ample
Table	Target Symbols	Paths
Т	401^{-1} 201^{-1}	$(401^{-1}) (201^{-1})$

[0290] Since each of the source tables S_1 and S_2 suffice to generate the target table T, the desired SQL is a union of a query involving S_1 alone and a query involving S_2 alone. Specifically, based on the paths given in Table LXXXV, the desired SQL query is:

INSERT INTO T (SELECT	(Emp_Name, Emp_Division, Emp_Tel_No)
	S ₁ .Employee_Name AS Emp_Name,
	S ₁ .Employee_Division AS Emp_Division,
	S ₁ .Employee Tel# AS Emp Tel No
FROM	
	S ₁
UNION	-
SELECT	
	S2.Employee_Name AS Emp_Name,
	S ₂ .Employee_Division AS Emp_Division,
	S ₂ .Employee Tel# AS Emp_Tel_No
FROM S_2 ;	

[0291] In general,

[0292] Rule 6: When one or more source tables contain source symbols sufficient to generate all of the target symbols, then each such source table must be used alone in an SQL query and the resultant queries joined. (Note that Rule 6 is consistent with Rule 5.)

[0293] When applied to the following sample source data, Tables LXXXVI and LXXXVII, the above SQL query produces the target data in Table LXXXVIII.

TABLE LXXXVI

Sample S	ource Table S ₁ for	Eighth Example	
Employee_Division	Employee_Tel#	Employee_Name	Room#
Engineering	113	Richard	10
SW	118	Adrian	4
Engineering	105	David	10

[0294]

TABLE LXXXVII		
Sample	e Source Table S ₂ for Eig	thth Example
Name	Employee_Tel	Division
Henry Robert William Richard	117 106 119 113	SW IT PdM Engineering

[0295]

TABLE LXXXVIII

Sample Target Table T for Eighth Example			
Emp_Name	Emp_Division	Emp_Tel_No	
Tom Adrian David Henry Robert	Engineering SW Engineering SW IT	113 118 105 117 106	
William	PdM	119	

A Ninth Example

Data Constraints

[0296] In a ninth example, a target table is of the following form:

TABLE LXXXIX

Target Table	T for Ninth Example
City	Temperature

[0297] Two source tables are given as follows:

Source Tal	ole S ₁ for Ninth Example
City	Temperature

[0298]

TABLE XCI

	Source Table S ₂ for Ninth Example	
City	C_Temperature	

[0299] The underlying ontology is illustrated in **FIG. 19**. The dotted portions of the ontology in **FIG. 19** are additional ontology structure that is transparent to the relational database schema. The properties temperature_in_Centrigade and temperature_in_Fahrenheit are related by the constraint:

Temperature_in_Centrigade(City) 5/9*(Temperature-_in_Fahrenheit(City)-32)

[0300] The unique properties of the ontology are:

TABLE XCII

Unique Properties within	Ontology for Ninth Example
Property	Property Index
name(City)	1

[0301] The mapping of the target schema into the ontology is as follows:

TABLE XCIII

Mapping from Target schema to Ontology for Ninth Example		
schema	Ontology	Property Index
T T.City T.Temperature	Class: City Property: name(City) Property: temperature_in_Centigrade (City)	1 2

[0302] The mapping of the source schema into the ontology is as follows:

TABLE XCIV

Mapping from Source schema to Ontology for Ninth Example

schema	Ontology	Property Index	
S ₁	Class: City		
S ₁ .City	Property: name(City)	1	
S ₁ .Temperature	Property:	3	
	temperature_in_Fahrenheit		
	(City)		
S ₂	Class: City		
S ₂ .City	Property: name(City)	1	
S ₂ .C_Temperature	Property:	2	
£ 1	temperature_in_Centrigade (City)		

[0303] The indices of the source properties are:

TABLE AC V		
	Source Symbols f	or Ninth Example
	Table	Source Symbols
	$egin{array}{c} \mathbf{S}_1 \ \mathbf{S}_2 \end{array}$	301 ⁻¹ 201 ⁻¹

TADLE YOU

[0304] The indices of the target properties, keyed on City are:

TABLE XCVI

	Target Symbols for N	Ninth Example
Table	Target Symbols	Paths
Т	201-1	⁵ / ₉ * ((301 ⁻¹)-32) (201 ⁻¹)

[0305] Since each of the source tables S_1 and S_2 suffice to generate the target table T, the desired SQL is a union of a query involving S_1 alone and a query involving S_2 alone. Specifically, based on the paths given in Table XCVI, the desired SQL query is:

INSERT IN	TO T(City, Temperature)
(SELECT	
	S ₁ .City AS City,
	5/9 * (S ₁ .Temperature - 32) AS Temperature
FROM	
	S ₁
UNION	
SELECT	
	S2.City AS City, S2.Temperature AS Temperature
FROM	
	\mathbf{S}_{2});
	$S_{2});$

[0306] In general,

[0307] Rule 7: When a target symbol can be expressed in terms of one or more source symbols by a dependency constraint then such constraint must appear in the list of target symbols.

[0308] When applied to the following sample source data, Tables XCVII and XCVIII, the above SQL query produces the target data in Table XCIX.

Sample Source Tab	Sample Source Table S ₁ for Ninth Example	
City	Temperature	
New York Phoenix Anchorage Boston	78 92 36 72	

22

[0309]

Tel Aviv London

TABLE XCVIII		
Sample Source Table S2 for Ninth Example		
City	C_Temperature	
Moscow	12	
Brussels	23	

[0310]

TABLE XCIX

Sample Target Table T for Ninth Example		
City	Temperature	
New York	25.5	
Phoenix	33.3	
Anchorage	2.22	
Boston	22.2	
Moscow	12	
Brussels	23	
Tel Aviv	32	
London	16	

A Tenth Example

Pricing

[0311] In a tenth example, a target table is of the following form:

TABLE C

Target Table T for Tenth Example			
	Product	Price	

[0312] Two source tables are given as follows:

TABLE CI

Source Table S ₁ for Tenth Example		th Example
SK	U	Cost

[0313]

TABLE CII

Source Table S_2 for Tenth Example		h Example
	Item	Margin

[0314] The underlying ontology is illustrated in **FIG. 20**. The dotted portions of the ontology in **FIG. 20** are additional ontology structure that is transparent to the relational database schema. The properties price, cost_of_production and margin are related by the constraint:

price(Product)=

(Draduat)

cost_of_production(Product)*margin(Product).

32

[0315] The unique properties of the ontology are:

Unique Properties within	Ontology for Tenth Example
Property	Property Index
SKU(Product)	1

TABLE CIII

[0316] The mapping of the target schema into the ontology is as follows:

TABLE CIV

Mapping fr	om Target schema to Ontology f	for Tenth Example
schema	Ontology	Property Index
Т	Class: Product	
T.Product	Property: SKU(Product)	1
T.Price	Property: price(Product)	4

[0317] The mapping of the source schema into the ontology is as follows:

TABLE CV

Mapping from Source schema to Ontology for Tenth Example		
schema	Ontology	Property Index
S ₁	Class: Product	
S ₁ .SKU	Property: SKU(Product)	1
S ₁ .Cost	Property: cost_of_production(Product)	2
S_2	Class: Product	
S_2 .Item	Property: SKU(Product)	1
S_2 .Margin	Property: margin(Product)	3

[0318] The indices of the source properties are:

TABLE CVI	
Source Symbols for Tenth Example	
Table	Source Symbols
$\overline{\begin{array}{c} S_1\\S_2\end{array}}$	201 ⁻¹ 301 ⁻¹

[0319] The indices of the target properties, keyed on Product are:

TABLE CVII

	Target Symbols for Tent	h Example
Table	Target Symbols	Paths
Т	4o1 ⁻¹	(201 ⁻¹) * (301 ⁻¹)

[0320] Based on the paths given in Table CVII, the desired SQL query is:

INSERT INTO	T(Product, Price)
EDOM	$\rm S_1.SKU~AS~Product,~(S_1.Cost)~*~(S_2.Margin)~AS~Price$
FROM	S ₁ , S ₂
WHERE	S_2 .Item = S_1 .SKU);

[0321] When applied to the following sample source data, Tables CVIII and CVIX, the above SQL query produces the target data in Table CX.

TAB	LE CVIII
Sample Source Ta	ble S ₁ for Tenth Example
SKU	Cost
123	2.2
234	3.3
345	4.4
456	5.5

TABLE CIX Sample Source Table S2 for Tenth Example Item Margin 1.2 123 234 1.1345 1.04 456 1.3

[0323]

TABLE CX

Sample Target Tal	ole T for Tenth Example	
 Product	Price	
123	2.86	
234	3.96	
345	4.84	
456	5.72	

An Eleventh Example

String Concatenation

[0324] In an eleventh example, a target table is of the following form:

TABLE CXI

Target Table T	for Eleventh Example
ID#	Full_Name

[0322]

[0325] One source table is given as follows:

TABLE CXII

	Source Table S for Elevent	h Example
ID#	First_Name	Last_Name

[0326] The underlying ontology is illustrated in FIG. 21. The dotted portions of the ontology in FIG. 21 a re additional ontology structure that is transparent to the relational database schema. The properties full_name, first-_name and last_name are related by the constraint:

full_name(Person)= first_name(Person)||last_name(Person),

[0327] where \parallel denotes string concatenation.

[0328] The unique properties of the ontology are:

TABLE CXIII

Property Property Index	Unique Properties within On	tology for Eleventh Example	
	Property	Property Index	
ID#(Product) 1	ID#(Product)	1	

[0329] The mapping of the target schema into the ontology is as follows:

	TABLE CXIV	
Mapping fr	om Target schema to Ontology for El	eventh Example
schema	Ontology	Property Index
Т	Class: Person	
T.ID#	Property: ID#(Person)	1
T.Full_Name	Property: full_name(Person)	4

[0330] The mapping of the source schema into the ontology is as follows:

TABLE CXV

Mapping from S	ource schema to Ontology for Ele-	venth Example
schema	Ontology	Property Index
S S.ID# S.First_Name S.Last_Name	Class: Person Property: ID#(Person) Property: first_name(Person) Property: last_name(Person)	1 2 3

[0331] The indices of the source properties are:

TABLE CXVI

Source Symbols fo	r Eleventh Example
Table	Source Symbols
S	201 ⁻¹ 201 ⁻¹

[0332] The indices of the target properties, keyed on ID# are:

TABLE CXVII

	Target Symbols for Ele	eventh Example
Table	Target Symbols	Paths
Т	4o1 ⁻¹	$(201^{-1}) (301^{-1})$

[0333] Based on the paths given in Table CXVII, the desired SQL query is:

INSERT IN (SELECT	TO T(ID#, Full_Name)
	S.ID# AS ID#, (S.First_Name) (S.Last_Name) AS Full_Name
FROM	S);

[0334] When applied to the following sample source data, Table CXVIII, the above SQL query produces the target data in Table CXIX.

TABLE CXVIII

Sample Source Table S for Eleventh Example		
ID#	First_Name	Last_Name
123	Timothy	Smith
234	Janet	Ferguson
345	Ronald	Thompson
456	Marie	Baker
567	Adrian	Clark

[0335]

TABLE CXIX

Sample Target Table T for Eleventh Example

ID#	Full_Name
123	Timothy Smith
234	Janet Ferguson
345	Ronald Thompson
456	Marie Baker
567	Adrian Clark

A Twelfth Example

Books→Documents

[0336] A source XML schema for books is given by:

<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLschema">

<xs:element name="book" type="Book"/>

<xs:complexType name="Book">

<xs:sequence>

<xs:element name="name" type="xs:string"/>

<xs:element name="author" type="Author"/>
-continued

</xs:sequence> </xs:complexType> <xs:complexType name="Author"> <xs:attribute name="name"/> </xs:complexType> </xs:schema>

[0337] A target XML schema for documents is given by:

<?xml version="1.0" encoding="UTF-8"?> <xs:schema xmlns:xs="http://www.w3.org/2001/XMLschema"> <xs:celement name="document" type="Document"/> <xs:complexType name="Document"> <xs:all> <xs:all> <xs:selement name="writer" type="xs:string"/> </xs:all> <xs:attribute name="title"/> </xs:complexType> </xs:complexType>

[0338] A common ontology model for the source and target XML schema is illustrated in **FIG. 22**. A mapping of the source XML schema into the ontology model is given by:

TABLE CXX

Mapping from for Twelfth	, 	
schema	Ontology	Property Index
complexType: book	Class: Book	
element: book/name/text()	Property: name(Book)	1
element: book/author	Property: author(Book)	2
complexType: author	Class: Person	
element: author/@name	Property: name(Person)	3

[0339] A mapping of the target XML schema into the ontology model is given by:

TABLE CXXI

Mapping from Target schema to Ontology for Twelfth Example		
schema	Ontology	Property Index
complexType: document	Class: Book	
element: document/writer/text()	Property: name(author(Book))	302
attribute: document/@title	Property: name(Book)	1

[0340] Tables CXX and CXXI use XPath notation to designate XSL elements and attributes.

[0341] Based on Tables CXX and CXXI, an XSLT transformation that maps XML documents that conform to the source schema to corresponding documents that conform to the target schema should accomplish the following tasks:

- [0342] 1. document/@title \leq book/name/text()
- [0343] 2. document/writer/text()与book/author/ @name

[0344] Such a transformation is given by:

<?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet version="1.0" xmlns:xsl= "http://www.w3.org/1999/XSL/Transform"> <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/> <xsl:template match="/"> <document> <xsl:for-each select=".//book[position()=1]"> <xsl:attribute name="title"> <xsl:value-of select="name()"/> </xsl:attribute> <xsl:element name="writer"> <xsl:value-of select="author/@name" /> </xsl:element> </xsl:for-each> </document> </xsl:template> </xsl:stylesheet>

A Thirteenth Example

Books→Documents

[0345] A source XML schema for books is given by:

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLschema">
</ss:schema xmlns:xs="http://www.w3.org/2001/XMLschema">
</ss:schema xmlns:xs="http://www.w3.org/2001/XMLschema">
</sss:schema xmlns:xs="http://www.w3.org/2001/XMLschema">
</sss:schema xmlns:xs="http://www.w3.org/2001/XMLschema">
</sss:schema xmlns:xs="http://www.w3.org/2001/XMLschema">
</sssichema xmlns:xs="http://www.w3.org/2001/XMLschema">
</sssichema xmlns:xs="http://www.w3.org/2001/XMLschema">
</sssichema xmlns:xs="http://www.w3.org/2001/XMLschema">
</sssichema xmlns:xs="http://www.w3.org/2001/XMLschema">
</ssichema xmlns:xs="http://www.w3.org/2001/XMLschema">
</ssichema xmlns:xs="http://www.w3.org/2001/XMLschema">
</ssichema xmlns:xs="http://www.w3.org/2001/XMLschema">
</ssichema xmlns:xs="http://www.w3.org/2001/XMLschema">
</ssichema xmlns:xs="http://www.w3.org/2001/XMLschema">
</schema xmlns:xs:attp://www.w3.org/2001/XMLschema">
</schema xmlos:</schema xmlos:</

[0346] A target XML schema for documents is given by:

xml version="1.0" encoding="UTF-8"? <xs:schema xmlns:xs="http://www.w3.org/2001/XMLschema"></xs:schema>
<xs:element name="document" type="Document"></xs:element>
<xs:complextype name="Document"></xs:complextype>
<xs:choice></xs:choice>
<xs:element <="" minoccurs="1" name="writer" td="" type="xs:string"></xs:element>
maxOccurs="unbounded"/>
<xs:element name="title" type="xs:string"></xs:element>
<xs:element name="ISBN" type="xs:string"></xs:element>

[0347] A common ontology model for the source and target XML schema is illustrated in FIG. 23. A mapping of the source XML schema into the ontology model is given by

Table CXVIII above. A mapping of the target XML schema into the ontology model is given by:

TABLE CXXII

schema	Ontology	<u>Property</u> Index
complexType: document element: document/writer/text()	Class: Book Property: name(author(Book))	302
element: document/title/text()	Property: name(Book)	1
element: document/ISBN/text()	Property: ISBN(Book)	4

[0348] Based on Tables CXX and CXXII, an XSLT transformation that maps XML documents that conform to the source schema to corresponding documents that conform to the target schema should accomplish the following tasks:

- [0349] 1. document/title/text() \leq book/name/text()
- [0350] 2. document/writer/text()与book/author/ @name
- [0351] Such a transformation is given by:

xml version="1.0" encoding="UTF-8"?
<xsl:stylesheet version="1.0" xmlns:xsl="</td"></xsl:stylesheet>
"http://www.w3.org/1999/XSL/Transform">
<xsl:output <="" encoding="UTF-8" method="xml" td="" version="1.0"></xsl:output>
indent="yes"/>
<xsl:template match="/"></xsl:template>
<document></document>
<xsl:apply-templates select="book"></xsl:apply-templates>
<xsl:template match="book"></xsl:template>
<xsl:choose></xsl:choose>
<xsl:when test="author"></xsl:when>
<xsl:for-each select="author"></xsl:for-each>
<xsl:element name="writer"></xsl:element>
<xsl:value-of select="@name"></xsl:value-of>
<xsl:when test="name"></xsl:when>
<xsl:element name="title"></xsl:element>
<xsl:value-of select="name/text()"></xsl:value-of>

A Fourteenth Example

Document Storage

[0352] A source XML schema for books is given by:

<?xml version="1.0" encoding="UTF-8"?> <xs:schema xmlns:xs="http://www.w3.org/2001/XMLschema"> <xs:element name="library" type="Library"/> <xs:complexType name="Library">

-continued

<xs:sequence></xs:sequence>
<xs:element <="" name="source" td="" type="Source"></xs:element>
minOccurs="0" maxOccurs="unbounded"/>
<xs:complextype name="Source"></xs:complextype>
<xs:sequence></xs:sequence>
<xs:element <="" name="review" td="" type="Review"></xs:element>
minOccurs="0" maxOccurs="unbounded"/>
<xs:element <="" name="article" td="" type="Article"></xs:element>
minOccurs="0" maxOccurs="unbounded"/>
<xs:element <="" name="letter" td="" type="Letter"></xs:element>
minOccurs="0" maxOccurs="unbounded"/>
<xs:complextype name="Review"></xs:complextype>
<xs:sequence></xs:sequence>
<xs:element <="" name="author" td="" type="xs:string"></xs:element>
minOccurs="0" maxOccurs="unbounded"/>
<xs:attribute name="title"></xs:attribute>
<xs:complextype name="Article"></xs:complextype>
<xs:sequence></xs:sequence>
<xs:element <="" name="writer" td="" type="xs:string"></xs:element>
minOccurs="0" maxOccurs="unbounded"/>
<xs:attribute name="name"></xs:attribute>
<xs:complextype name="Letter"></xs:complextype>
<xs:sequence></xs:sequence>
<xs:element <="" name="sender" td="" type="xs:string"></xs:element>
minOccurs="0" maxOccurs="unbounded"/>
<pre></pre> <pre></pre> <pre>//> </pre> <pre>//> </pre>
versite name - name />
<xs.attribute name="subject"></xs.attribute>
<pre><ss.autiouc lidilic="leccivel"></ss.autiouc> </pre>

[0353] A first target XML schema for documents is given by:

```
<?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"/>
```

</ts:sequence> <xs:attribute name="title"/> </ts:complexType> <xs:complexType name="Letter"> <xs:complexType name="Letter"> <xs:sequence> <xs:element name="author" type="xs:string" minOccurs="0" maxOccurs="unbounded"/> </ts:sequence> <xs:attribute name="name"/> <xs:attribute name="subject"/> <xs:attribute name="receiver"/> </ts:complexType> </ts:complexType>

[0354] A common ontology model for the source and first target XML schema is illustrated in FIG. 24. A mapping of the source XML schema into the ontology model is given by:

TABLE CXXIII

Mapping from Source schema to Ontology for Fourteenth Example

schema	Ontology	Property Index
complexType: review	Class: Document	
element: review/author/text()	Property: author(Document)	1
attribute: review/@title	Property: title(Document)	2
complexType: article	Class: Document	
element: article/writer/text()	Property: author(Document)	1
attribute: article/@name	Property: title(Document)	2
complexType: letter	Class: Letter	
	(inherits from Document)	
element: letter/sender/text()	Property: author(Letter)	1
attribute: letter/@name	Property: title(Letter)	2
attribute: letter/@subject	Property: subject(Letter)	3
attribute: letter/@receiver	Property: receiver(Letter)	4
complexType: source	Class: Storage	
ComplexType: library	Container Class:	
1 51	set[Storage]	

[0355] A mapping of the first target XML schema into the ontology model is given by:

TABLE CXXIV

Mapping from First Target schema to Ontology for Fourteenth Example		
schema	Ontology	Property Index
complexType: document	Class: Document	
element: document/author/text()	Property: author(Document)	1
attribute: document/@title	Property: title(Document)	2
complexType: letter	Class: Letter	
	(inherits from Document)	
element: letter/author/text()	Property: author(Letter)	1
attribute: letter/@name	Property: title(Letter)	2
attribute: letter/@subject	Property: subject(Letter)	3
attribute: letter/@receiver	Property: receiver(Letter)	4
complexType: storage	Class: Storage	
element: storage/articles	Property: articles(Storage)	9
element: storage/reviews	Property: reviews(Storage)	10
element: storage/letters	Property: letters(Storage)	11

[0356] Based on Tables CXXIII and CXXIV, an XSLT transformation that maps XML documents that conform to

the source schema to corresponding documents that conform to the target schema should accomplish the following tasks:

- [0357] 1. storage slibrary
- [0358] 2. letter/author/text() \u2255 letter/sender/text()
- [0359] Such a transformation is given by:

<?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <xsl:template match="/"> <xsl:apply-templates select="/library"/> </xsl:template> <xsl:template match="/library"> <storage> <articles> <rul><xsl:apply-templates select="source[not(letter)]/article | source[not(review)]/article"/> </articles> <reviews> <xsl: apply-templates select="source[not(letter)]/review"/> </reviews> <letters> <xsl:apply-templates select="source[not(review)]/letter"/> </letters> </storage> </xsl:template> <xsl:template match="article"> <article> <xsl:attribute name="title"><xsl:value-of select="@name"/></xsl:attribute> <xsl:apply-templates select="writer"/> </article> </xsl:template> <xsl:template match="review"> <review> <xsl:attribute name="title"><xsl:value-of select="@title"/></xsl:attribute> <xsl:apply-templates select="author"/> </review> </xsl:template> <xsl:template match="letter"> <review> <xsl:attribute name="name"><xsl:value-of select="@name"/></xsl:attribute> <xsl:attribute name="subject"><xsl:value-of select="@subject"/></xsl:attribute> <xsl:attribute name="receiver"><xsl:value-of select="@receiver"/></xsl:attribute> <xsl:apply-templates select="sender"/> </review> </xsl:template> <xsl:template match="article/writer | review/author | letter/sender"> <author> <xsl:value-of select="text()"/> </author> </xsl:template> </xsl:stylesheet>

[0360] A second target XML schema for documents is given by:

xml version="1.0" encoding="UTF-8"?
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLschema"></xs:schema>
<xs:element name="storage" type="Storage"></xs:element>
<xs:complextype name="Storage"></xs:complextype>
<xs:sequence></xs:sequence>
<xs:element name="books" type="Books"></xs:element>
<xs:element name="magazines" type="Magazines"></xs:element>

-continued

<pre> </pre> <pre> cvs:complexType name="Books"> </pre>
<pre><xs:complextype name="books"> </xs:complextype></pre>
<pre><xs:element name="articles" type="Documents"></xs:element></pre>
<pre>cys:element name="reviews" type="Documents"/> cys:element name="reviews" type="Documents"/></pre>
/vs.seculences
<xs:complextype></xs:complextype>
<pre><xs:complextypenance magazines=""> </xs:complextypenance></pre>
<pre><xs:sequences <="" pre=""> <pre>/> </pre> <pre>//> </pre> <pre>//> </pre> <pre>//> </pre></xs:sequences></pre>
<pre><xs:element name="letters" type="Letters"></xs:element></pre>
<xs:complextype name="Documents"></xs:complextype>
<xs:sequence></xs:sequence>
<pre><xs:element <="" name="document" pre="" type="Document"></xs:element></pre>
minOccurs="0"
maxOccurs="unbounded"/>
<xs:complextype name="Letters"></xs:complextype>
<xs:sequence></xs:sequence>
<xs:element <="" name="letter" td="" type="Letter"></xs:element>
minOccurs="0" maxOccurs="unbounded"/>
<xs:complextype name="Document"></xs:complextype>
<xs:sequence></xs:sequence>
<xs:element <="" name="author" td="" type="xs:string"></xs:element>
minOccurs="0" maxOccurs="unbounded"/>
<xs:attribute name="title"></xs:attribute>
<xs:complextype name="Letter"></xs:complextype>
<xs:sequence></xs:sequence>
<xs:element <="" name="author" td="" type="xs:string"></xs:element>
minOccurs="0" maxOccurs="unbounded"/>
<xs:attribute name="name"></xs:attribute>
<xs:attribute name="subject"></xs:attribute>
<xs:attribute name="receiver"></xs:attribute>

[0361] A mapping of the second target XML schema into the ontology model is given by:

TABLE CXXV

Mapping from Second Target schema to Ontology for Fourteenth Example		
schema	Ontology	Property Index
complexType: document	Class: Document	
element:	Property: author(Document)	1
document/author/text()		
attribute:	Property: title(Document)	2
document/@title		
complexType: letter	Class: Letter	
	(inherits from Document)	
element:	Property: author(Letter)	1
letter/author/text()		
attribute: letter/@name	Property: title(Letter)	2
attribute: letter/@subject	Property: subject(Letter)	3
attribute: letter/@receiver	Property: receiver(Letter)	4
complexType: storage	Class: Storage	
element: storage/books	Property: books(Storage)	12
element: storage/magazines	Property: magazines(Storage)	13
complexType: book	Class: Book	
element: book/articles	Property: articles(Book)	5
element: book/reviews	Property: reviews(Book)	6

TABLE CXXV-continued

Mapping to Ontole		
schema	Ontology	Property Index
complexType: magazine	Class: Magazine	
element: magazine/articles	Property: articles(Magazine)	7
element: magazine/letters	Property: letters(Magazine)	8

[0362] Based on Tables CXXIII and CXXV, an XSLT transformation that maps XML documents that conform to the source schema to corresponding documents that conform to the target schema should accomplish the following tasks:

- [0363] 1. storage ≤ library
- [0364] 2. letter/author/text() \u2254 letter/sender/text()
- **[0365]** Such a transformation is given by:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
    <xsl:apply-templates select="/library"/>
</xsl:template>
<xsl:template match="/library">
    <storage>
          <books>
              <articles>
                   <xsl:apply-templates select=
                    "source[not(letter)]/article"/>
              </articles>
              <reviews>
                   <xsl:apply-templates select=
                    "source[not(letter)]/review"/>
              </reviews>
         </books>
         <magazines>
              <articles>
                   <xsl:apply-templates select=
                    "source[not(review)]/article"/>
              </articles>
              <letters>
                   <xsl:apply-templates select=
                    "source[not(review)]/letter"/>
              </letters>
         </magazines>
    </storage>
</xsl:template>
<xsl:template match="article">
    <article>
          <xsl:attribute name="title"><xsl:value-of
         select="@name"/></xsl:attribute>
          <xsl:apply-templates select="writer"/>
    </article>
</xsl:template>
<xsl:template match="review">
     <review>
          <xsl:attribute name="title"><xsl:value-of
         select?="@title"/></xsl:attribute>
          <xsl:apply-templates select="author"/>
     </review>
</xsl:template>
<xsl:template match="letter">
    <review>
         <xsl:attribute name="name"><xsl:value-of
         select="@name"/></xsl:attribute>
          <xsl:attribute name="subject"><xsl:value-of
         select="@subject"/></xsl:attribute>
```

-continued

<xsl:attribute name="receiver"><xsl:value-of select="@receiver"/></xsl:attribute> <xsl:apply-templates select="sender"/> </review> </xsl:template> <xsl:template match="article/writer | review/author | letter/sender"> <author> <xsl:value-of select="text()"/> </author> </xsl:template> </xsl:stylesheet>

[0366] A third target XML schema for documents is given by:

<?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="article_from_books"</pre> type="AB" minOccurs="0" maxOccurs="unbounded"/> <xs:element name="article_from_magazines" type="AM" minOccurs="0" maxOccurs="unbounded"/> </xs:sequence> </xs:complexType> <xs:complexType name="AB"> <xs:sequence> <xs:element name="authors" type="xs:string" minOccurs="0" maxOccurs="unbounded"/> </xs:sequence> <xs:attribute name="title"/> </xs:complexType> <xs:complexType name="AM"> <xs:sequence> <xs:element name="writers" type="xs:string" minOccurs="0" maxOccurs="unbounded"/> </xs:sequence> <xs:attribute name="name"/> </xs:complexType> </xs:schema>

[0367] A mapping of the third target XML schema into the ontology model is given by:

TABLE CXXVI

Mapping from Third Target schema to Ontology for Fourteenth Example			
schema	(Ontology	Property Index
complexType: AE	3 (Class: Document	
element: AB/auth	or/text() l	Property: author(Document)	1
attribute: AB/@tit	ile l	Property: title(Document) Class: Document	2
element: AM/writ	er/text()]	Property: author(Document)	1
attribute: AM/@ti complexType: sto	tle l rage d	Property: title(Document) Complex Class: set[Document] × set[Document]	2

[0368] Based on Tables CXXIII and CXXVI, an XSLT transformation that maps XML documents that conform to

the source schema to corresponding documents that conform to the target schema should accomplish the following tasks:

[0369] 1. storage ≒library

- [0370] 2. letter/author/text() \u2254 letter/sender/text()
- [0371] Such a transformation is given by:

<?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <xsl:template match="/"> <xsl:apply-templates select="/library"/> </xsl:template> <xsl:template match="/library"> <storage> <xsl:apply-templates select="source[not(letter)]/article"mode="AB"/> <xsl:apply-templates select="source[not(review)]/article"mode="AM"/> </storage> </xsl:template> <xsl:template match="article" mode="AB"> <article_from_books> <xsl:attribute name="title"><xsl:value-of select="@name"/></xsl:attribute> <xsl:apply-templates select="writer" mode="AB"/> </article_from_books> </xsl:template> <xsl:template match="article" mode="AM"> <article_from_magazines> <xsl:attribute name="name"><xsl:value-of select="@name"/></xsl:attribute> <xsl:apply-templates select="writer" mode="AM"/> </article_from_magazines> </xsl:template> <xsl:template match="article/writer" mode="AB"> <author: <xsl:value-of select="text()"/> </author> </xsl:template> <xsl:template match="article/writer" mode="AM"> <writer> <xsl:value-of select="text()"/> </writer> </xsl:template> </xsl:stylesheet>

A Fifteenth Example

String Conversion

[0372] A source XML schema for people is given by:

xml version="1.0" encoding="UTF-8"?
<xs:schema xmins:xs="http://www.w5.org/2001/XMLschema</td"></xs:schema>
elementFormDefault= qualified
attributeFormDefault="unqualified">
<xs:element name="Person" type="Person"></xs:element>
<xs:complextype name="Person"></xs:complextype>
<xs:sequence></xs:sequence>
<xs:element name="name" type="xs:string"></xs:element>
name expected input in format</td
firstName#LastName>
<xs:element name="ID" type="xs:string"></xs:element>
ID expected input in format XXXXXXXXX-X
<xs:element name="age" type="xs:string"></xs:element>
age expected input in exponential form XXXeX

[0373] A target XML schema for people is given by:

FirstName -->

-continued

[0374] An XSLT transformation that maps the source schema into the target schema is given by:

<?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/> <xsl:template match="/"> <Person> <xsl:for-each select="Person"> <xsl:element name="name"> <xsl:value-of select= "concat(substring-after(name, '#'), ', substring-before(name, '#'))"/> </xsl:element> <xsl:element name="ID"> <xsl:variable name="plainID" select= "concat(substring-before(ID/text(),'-'),substring-after(ID/text(),'-'))"/> <xsl:value-of select= "concat('12',substring(\$plainID,1,2),'-',substring(\$plainID,3),'3E')"/> </xsl:element> <xsl:element name="age"> <xsl:call-template name="exponentiate"> <xsl:with-param name="power" select="substring-after(age,'e')"/> <xsl:with-param name="digit" select="substring-before(age,'e')"/> <xsl:with-param name="ten" select="1"/> </xsl:call-template> </xsl:element> </xsl:for-each> </Person> </xsl:template> <xsl:template name="exponentiate"> <xsl:param name="power"/> <xsl:param name="digit"/> <xsl:param name="ten"/> <xsl:choose> <xsl:when test="\$power > 0"> <xsl:call-template name="exponentiate"> <xsl:with-param name="power" select="\$power - 1"/> <xsl:with-param name="digit" select="\$digit"/> <xsl:with-param name="ten" select="\$ten * 10"/> </xsl:call-template> </xsl:when> <xsl:when test="\$power < 0"> <xsl: call-template name="exponentiate"> <xsl:with-param name="power" select="\$power + 1"/> <xsl:with-param name="digit" select="\$digit"/> <xsl:with-param name="ten" select="\$ten div 10"/> </xsl:call-template> </xsl:when> <xsl:otherwise> <xsl:value-of select="format-number(\$digit * \$ten,',###.###") "/> </xsl:otherwise> </xsl:choose> </xsl:template> </xsl:stylesheet>

A Sixteenth Example

String Conversion

[0375] A source XML schema for people is given by:

<?xml version="1.0" encoding="UTF-8"?> <xs:schema xmlns:xs=http://www.w3.org/2001/XMLschema elementFormDefault="qualified" attributeFormDefault="unqualified"> <xs:element name="Person" type="Person"/>

-continued

<Xs:complexType name="Person"> <xs:sequence> <xs:element name="name" type="xs:string"/> <xs:element name="homeTown" type="xs:string"/> </xs:sequence> <xs:attribute name="dog_name"/> </xs:complexType> </xs:schema>

[0376] A target XML schema for people is given by:

<?xml version="1.0" encoding="UTF-8"?> <xs:schema xmlns:xs="http://www.w3.org/2001/XMLschema" elementFormDefault="qualified" attributeFormDefault="unqualified"> <xs:element name="Person" type="Person"/> <xs:complexType name="Person"> <xs:complexType name="Person"> <xs:sequence> <xs:element name="name" type="xs:string"/> <xs:element name="homeTown" type="xs:string"/> </xs:sequence> <xs:attribute name="dog_name"/> </xs:complexType> </xs:complexType> </xs:cohema>

[0377] An XSLT transformation that maps the source schema into the target schema is given by:

xml version="1.0" encoding="UTF-8"? <xsl:stylesheet <="" th="" version="1.0"></xsl:stylesheet>		
mlns:xsl="http://www.w3.org/1999/XSL/Transform">		
<xsl:output <="" method="xml" td="" version=" 1.0"></xsl:output>		
encoding="UTF-8" indent="yes"/>		
<xsl:template match="/"></xsl:template>		
<person></person>		
<xsl:for-each select="Person"></xsl:for-each>		
<xsl:attribute name="dog"></xsl:attribute>		
<xsl:value-of select="@dog_name"></xsl:value-of>		
<xsl: element="" name="name"></xsl:>		
<xsl:value-of select="name/text()"></xsl:value-of>		
<xsl:elementname="indexofcarstring_caseinsensitive"></xsl:elementname="indexofcarstring_caseinsensitive">		
<pre><xsl:variable <="" name="case_neutral" pre=""></xsl:variable></pre>		
select="translate(name,		
'ABCDEFGHIJKLMNOPQRSTUVWXYZ',		
'abcdefghijklmnopqrstuvwxyz')"/>		
<xsl:value-of select="</td"></xsl:value-of>		
"string-length(substring-before(\$case neutral.		
(car')) - 1"/>		
<xsl:element name="indexOfcarString CaseSensitive"></xsl:element>		
<pre><xsl:value-of))<="" pre="" select="string-length(substring-before(name."></xsl:value-of></pre>		
(car')) - 1"/>		
<pre></pre> <pre></pre> <pre></pre> <pre>// "> </pre>		
<pre></pre> <pre><</pre>		
(velue)ements		

A Seventeenth Example

Library→Storage

[0378] A source XML schema for libraries is given by:

xml version="1.0" encoding="UTF-8"? <xs:schema xmlns:xs="http://www.w3.org/2001/XMLschema"> <xs:element name="library" type="Library"></xs:element> <xs:complextype name="Library"></xs:complextype></xs:schema>
<pre><xs:sequence> </xs:sequence></pre> <pre><rue> </rue></pre> <pre><rue> </rue></pre> <pre></pre> <pre< td=""></pre<>
minOccurs="0" maxOccurs="unbounded"/>
<xs:complextype name="Book"></xs:complextype>
<xs:sequence></xs:sequence>
<xs:element name="name" type="xs:string"></xs:element>
<pre><xs:element <="" name="author" pre="" type="Author"></xs:element></pre>
maxOccurs="unbounded"/>
<xs:complextype name="Author"></xs:complextype>
<xs:attribute name="name"></xs:attribute>

[0379] A target XML schema for storage is given by:

<?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="document" type="Document" minOccurs="0" maxOccurs="unbounded"/> </xs:sequence> </xs:complexType> <xs:complexType name="Document"> <xs:sequence> <xs:element name="writer" type="xs:string" maxOccurs="unbounded"/> </xs:sequence> <xs:attribute name="title"/> </xs:complexType> </xs:schema>

[0380] A common ontology model for the source and target XML schema is illustrated in **FIG. 22**. A mapping of the source XML schema into the ontology model is given by Table CXX, with an additional correspondence between the complexType and the container class set[Book]. A mapping of the target XML schema into the ontology model is given by Table CXXI, with an additional correspondence between the complexType storage and the container class set{Book].

[0381] Based on Tables CXX and CXXI, an XSLT transformation that maps XML documents that conform to the source schema to corresponding documents that conform to the target schema should accomplish the following tasks:

[0382] 1. document/@title \Rightarrow book/name/text()

[0383] 2. document/writer/text()与book/author/ @name [0384] Such a transformation is given by:

<?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/> <xsl:template match="/"> <storage> <xsl:for-each select=".//library"> <xsl:for-each select="book"> <document> <xsl:attribute name="title"> <xsl:value-of select="name"/> </xsl:attribute> <writer> <xsl:for-each select="author/@name"> <xsl: value-of select="."/> </xsl:for-each> </writer> </document> </xsl:for-each> </xsl:for-each> </storage> </xsl:template> </xsl:stylesheet>

An Eighteenth Example

Change Case

[0385] A source XML schema for plain text is given by:

xml version="1.0" encoding="UTF-8"?
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLschema</td"></xs:schema>
elementFormDefault="qualified"
attributeFormDefault="unqualified">
<xs:element name="Person" type="Person"></xs:element>
<xs:complextype name="Person"></xs:complextype>
<xs:sequence></xs:sequence>
<pre><xs:element name="name" type="xs:string"></xs:element></pre>
<xs:element name="homeTown" type="xs:string"></xs:element>

[0386] A target XML schema for case sensitive text is given by:

xml version="1.0" encoding="UTF-8"? <xs:schema <="" th="" xmlns:xs="http://www.w3.org/2001/XMLschema"></xs:schema>
elementFormDefault="qualified"
attributeFormDefault="unqualified">
<xs:element name="Person" type="Person"></xs:element>
<xs:complextype name="Person"></xs:complextype>
<xs:sequence></xs:sequence>
<xs:element name="name" type="xs:string"></xs:element>
<xs:element name="homeTown" type="xs:string"></xs:element>

[0387] An XSLT transformation that maps the source schema into the target schema is given by:

[0389] A target XML schema for a list of numbers is given by:

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLschema"

<?xml version="1.0" encoding="UTF-8"?>

xml version="1.0" encoding="UTF-8"?		
<xsl: <="" stylesheet="" th="" version="1.0"></xsl:>		
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">		
<xsl:output <="" encoding="UTF-8" method="xml" th="" version="1.0"></xsl:output>		
indent="yes"/>		
<xsl:template match="/"></xsl:template>		
<person></person>		
<xsl:for-each select="Person"></xsl:for-each>		
<xsl:element name="low_name"></xsl:element>		
<xsl:value-of select="translate(name,</th></tr><tr><th>'ABCDEFGHIJKLMNOPQRSTUVWXYZ',</th></tr><tr><th>'abcdefghijklmnopqrstuvwxyz')"></xsl:value-of>		
<xsl:element name="upper_homeTown"></xsl:element>		
<xsl:value-of 1.0"="" ?="" encoding="UTF-8" select="translate(homeTown,</th></tr><tr><th>'abcdefghijklmnopqrstuvwxyz',</th></tr><tr><th><pre>'ABCDEFGHIJKLMNOPQRSTU-
VWXYZ')'/></pre></th></tr><tr><th></xsl:element></th></tr><tr><td></xsl:for-each</td></tr><tr><td></Person></td></tr><tr><td></xsl:template></td></tr><tr><td></xsl:stylesheet></td></tr><tr><th></th></tr><tr><td></td></tr><tr><th>An Nineteenth Example</th></tr><tr><td>Number Manipulation</td></tr><tr><th>[0388] A source XML schema for list of numbers is given by:</th></tr><tr><td><?xml version="> <xs:schema <br="" xmlns:xs="http://www.w3.org/2001/XMLschema">elementFormDefault="qualified"</xs:schema></xsl:value-of>		
attributeronniberaun= unquanneu >		

</xs:complexType>

</xs:schema>

elementFormDefault="qualified" attributeFormDefault="unqualified"> <rs:element name="List_o_Numbers" type="NumList"/> <xs:complexType name="NumList"> <xs:sequence> <xs:element name="first_as_num" type="xs:decimal"/> <!-- first_as_num - take a string and return a numerical value. Exemplifies use of the operator value(string) --> <xs:element name="second_floor" <!--second_floor return type="xs:decimal"/> nearest integer less than number. Exemplifies use of the operator floor(number) --> <xs:element name="second_firstDecimal_floor" type="xs:decimal"/> <!-- second_firstDecimal_floor - return nearest first decimal place less than number. Exemplifies use of the operator floor(number, significance) --> <xs:element name="third_ceil" type="xs:decimal"/> <!- third_ceil - return nearest integer greater than number. Exemplifies use of the operator ceil(number) --> <xs:element name="third_secondDecimal_ceil" type="xs:decimal"/> <!-- third_secondDecimal_ceil - return nearest second decimal place greater than number. Exemplifies use of the operator cei(number, significance) --> <xs:element name="fourth_round" type="xs:decimal"/> <!--fourth_round - round the number in integers. Exemplifies use of the operator round(number) --> <xs:element name="fourth_thirdDecimal_round"</pre> type="xs:decimal"/> <!-- fourth_thirdDecimal_round - round the number up to third decimal. Exemplifies use of the operator round(number, significance) --> <xs:element name="fifth_roundToThousand"</pre> type="xs:decimal"/> <!-- fifth_roundToThousand - round the number up to nearest ten to the third. Exemplifies use of the operator roundToPower(number, power) --> <xs:element name="abs_sixth" type="xs:decimal"/> <!-- abs_sixth - return absolute value of number. Exemplifies use of operator abs(number) --> <xs:element name="seventh" type="xs:string" /> <!-- seventh - return number as string. Exemplifies use of operator string(number) --> </xs:sequence> </xs:complexType>

</xs:schema>

34

[0390] An XSLT transformation that maps the source schema into the target schema is given by:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
<xsl:template match="/">
     <List_o_Numbers>
         <xsl:for-each select="List_o_Numbers">
              <xsl:element name="first_as_num">
                   <xsl:value-of select="number(first)"/>
              </xsl:element>
                                   <!-- first_as_num - take a string and return a numerical value.
                                             Exemplifies use of the operator value(string) -->
              <xsl:element name="second_floor">
                   <xsl:value-of select="floor(second)"/>
                                   <!-- second_floor return nearest integer less than number.
              </xsl:element>
                                        Exemplifies use of the operator floor(number) -->
              <xsl:elementname="second_firstDecimal_floor">
                   <xsl:value-of select="floor(second*10) div 10"/>
              </xsl:element>
                                   <!-- second_firstDecimal_floor - return nearest first decimal
              place less than number. Exemplifies use of the operator floor(number, significance) -->
              <xsl: element name="third_ceil">
                   <xsl:value-of select="ceiling(third)"/>
              </xsl:element>
              <xsl: element name="third secondDecimal_ceil">
                   <xsl:value-of select="ceiling(third*100) div 100"/>
                                   <!-- third_ceil - return nearest integer greater than number.
              </xsl:element>
                                             Exemplifies use of the operator ceil(number) -->
              <xsl:element name="fourth_round">
                   <xsl:value-of select="round(fourth)"/>
                                   <!--fourth_round - round the number in integers
              </xsl:element>
              Exemplifies use of the operator round(number) --> <xsl:element name="fourth_thirdDecimal_round">
                   <xsl:value-of select="round(fourth*1000) div 1000" />
                                   <!-- fourth_thirdDecimal_round - round the number up to
              </xsl:element>
                       third decimal. Exemplifies use of the operator round(number, significance) -->
              <xsl:element name="fifth_roundToThousand">
                   <xsl:value-of select="round(fifth div 1000) * 1000" />
              </xsl:element>
                                   <!-- fifth_roundToThousand - round the number up to nearest
                   ten to the third. Exemplifies use of the operator roundToPower(number, power) -->
              <xsl:element name="abs_sixth">
                   <xsl:choose>
                        <xsl:when test="sixth < 0">
                        <xsl:value-of select="sixth * -1"/>
                        </xsl:when>
                        <xsl:otherwise>
                            <xsl:value-of select="sixth"/>
                        </xsl:otherwise>
                   </xsl:choose>
              </xsl:element>
                                    <!-- abs_sixth - return absolute value of number.
                                                      Exemplifies use of operator abs(number) -->
              <xsl:element name="seventh">
                   <xsl:value-of select="concat(' ',string(seventh),' ')"/>
              </xsl:element>
                                   <!-- seventh - return number as string.
                                                  Exemplifies use of operator string(number) -->
         </xsl:for-each>
     </List_o_Numbers>
</xsl:template>
</xsl:stylesheet>
```

A Twentieth Example

String Manipulation

[0391] A source XML schema for a person is given by:

attributeFormDefault="unqualified">

-continued

<xs:element name="Person" type="Person"/> <xs:complexType name="Person"> <xs:sequence> <xs:element name="name" type="xs:string"/> <xs:element name="homeTown" type="xs:string"/> </xs:sequence> <xs:attribute name="dog_name" /> </xs:complexType> </xs:complexType>

<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLschema" elementFormDefault="qualified"

[0392] A target XML schema for a person is given by:

the first letter of a town,

-continued

out of respect. This exemplifies use of the capital operator--> </xs:sequence> <xs:attribute name="dog_trim"/> <xs:attribute name="dog_length"/> <!-- dog_trim - keep your dog trim - no blank spaces in front or after the name. This exemplifies use of the trim operator --> <!--dog_length - gives the number of characters (in integers, not dog years) in your dog's name. This exemplifies use of the length(string) operator --> </xs:complexType> </xs:schema>

[0393] An XSLT transformation that maps the source schema into the target schema is given by:

<?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/> <xsl:template match="/"> <Person> <xsl:for-each select="Person"> <xsl:attribute name="dog_trim"> <xsl:value-of select="normalize-space(@dog_name)"/> </xsl:attribute> <xsl:attribute name="dog_length"> <xsl:value-of select="string-length(normalize-space(@dog_name))"/> </xsl:attribute> <!-- dog_trim - This exemplifies use of the trim operator --> <!--dog_length - This exemplifies use of the length(string) operator -> <xsl:element name="four_name"> <xsl:value-of select="substring(name,1, 4)"/> </xsl:element> <xsl:element name="capital_homeTown"> <xsl:value-of select="concat(translate(substring(normalizespace(homeTown),1,1), 'abcdefghijklrnnopqrstuvwxyz', 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'), substring(normalize-space(homeTown),2))" /> </xsl:element> <!-- four-Name. This exemplifies use of the substring(string,start,length) operator--> <!-- capital_hometown. This exemplifies use of the capital operator--> </xsl:for-each> </Person> </xsl:template> </xsl:stylesheet>

Temperature Conversion

[0394] A source XML schema for temperature in Fahrenheit is given by:

<?xml version="1.0" encoding="UTF-8"?> <xs:schema xmlns:xs="http://www.w3.org/2001/XMLschema" elementFormDefault="qualified" attributeFormDefault="unqualified"> <xs:element name="city" type="city"/> <xs:complexType name="city"> <xs:complexType name="city"> <xs:sequence> <xs:sequence> <xs:sequence> <xs:sequence> <xs:sequence> <xs:settribute name="name" /> </xs:complexType> </xs:schema>

[0395] A target XML schema for temperature in Centigrade is given by:

<?xml version="1.0" encoding="UTF-8"?> <xs:schema xmlns:xs="http://www.w3.org/2001/XMLschema" elementFormDefault="qualified" <xs:element name="town" type="town" /> <xs:complexType name="town"> <xs:complexType name="town"> <xs:sequence> <xs:sequence> <xs:element name="temperatureC" type="xs:string" /> </xs:complexType> <xs:attribute name="name" /> </xs:schema>

[0396] An XSLT transformation that maps the source schema into the target schema is given by:

<?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/> <xsl:template match="/"> <town> <xsl:for-each select="city"> <xsl:attribute name="name"> <xsl:value-of select="@name"/> </xsl:attribute> <xsl:element name="temperatureC"> <xsl:value-of select=select="floor((temperatureF - 32) * (5 div 9))" /> </xsl:element> </xsl:for-each> </town> </xsl:template> </xsl:stylesheet>

A Twenty-Second Example

Town with Books

[0397] A source XML schema for a town with books is given by:

<?xml version="1.0" encoding="UTF-8"?> <xs:schema xmlns:xs="http://www.w3.org/2001/XMLschema" elementFormDefault="qualified" attributeFormDefault="unqualified"> <xs:element name="town" type="Town" /> <xs:complexType name="Town"> <xs:sequence> <xs:element name="library' type="Library" minOccurs="0" maxOccurs="unbounded" /> </xs:sequence> <xs:attribute name="name" type="xs:string" /> </xs:complexType> <xs:complexType name="Library"> <xs:sequence> <xs:element name="book" type="Book" minOccurs="0" maxOccurs="unbounded" /> </xs:sequence> <xs:attribute name="name" type="xs:string" /> </xs:complexType> <xs:complexType name="Book"> <xs:sequence> <xs:element name="title" type="xs:string" /> <xs:element name="author_name" type="xs:string" minOccurs="1" maxOccurs="unbounded" /> </xs:sequence> </xs:complexType> </xs:schema>

[0398] A target XML schema for a list of books is given by:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLschema"
elementFormDefault="qualified"
                          attributeFormDefault="unqualified">
<xs:element name="list_of_books" type="books"/>
<xs:complexType name="books">
    <xs:sequence>
         <xs:element name="book"
        type="book" minOccurs="0"
        maxOccurs="unbounded" />
    </xs:sequence>
</xs:complexType>
<xs:complexType name="book">
    <xs:sequence>
         <xs:element name="title"
        type="xs:string" />
         <xs:element name="author_name"
        type="xs:string" minOccurs="1"
                          maxOccurs="unbounded" />
    </xs:sequence>
</xs:complexType>
</xs:schema>
```

[0399] A common ontology model for the source and target XML schema is illustrated in **FIG. 25**. A mapping of

TABLE CXXVII

Mapping from Sour for Twenty-	Mapping from Source schema to Ontology for Twenty-Second Example	
schema	Ontology	Property Index
complexType: book	Class: Book	
element: book/title/text()	Property: name(Book)	1
element: book/author_name/text() complexType: library	Property: author(Book) Class: Library	2
element: library/books	Container Class: set[Book]	5
element: library/name/text() complexType: town	Property: name(Library) Class: Town	6
element: town/libraries	Container Class: set[Library]	1
element: town/name/text()	Property: name(Town)	2

[0400] A mapping of the target XML schema into the ontology model is given by:

TABLE CXXVIII

-	Mapping from Target schema to Ontology for Twenty-Second Example		
schema		Ontology	Property Index
complexType: h element: book/t element: book/z element: list_o	oook itle/text() uuthor_name/text() f_books	Class: Book Property: name(Book) Property: author(Book) Set[Book]	1 2

[0401] Based on Tables CXXVII and CXXVIII, an XSLT transformation that maps XML documents that conform to the source schema to corresponding documents that conform to the target schema is given by:

xml versions"1.0" encoding="UTF-8"? <xsl:stylesheet <br="" version="1.0">xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <xsl:output <="" method="xml" th="" version="1.0"></xsl:output></xsl:stylesheet>
encoding="UTF-8" indent="yes"/>
<xsl:template match="/"></xsl:template>
<books></books>
<xsl:for-each select=".//book"></xsl:for-each>
<book></book>
<xsl:element name="title"></xsl:element>
<xsl:value-of select="title/text()"></xsl:value-of>
<pre><xsl:for-each select="author_name"></xsl:for-each></pre>
~Ashistylesheets

A Twenty-Third Example

Town with Books

[0402] A source XML schema for a town is given by:

xml version="1.0" encoding="UTF-8"?
<xs:schema <br="" xmlns:xs="http://www.w5.org/2001/XMLschema">elementFormDefault="oualified"</xs:schema>
attributeFormDefault="unqualified">
<pre><xs:element name="town" type="Town"></xs:element></pre>
<pre><ws:complextype name="Town"></ws:complextype></pre>
<pre><vc:seculence></vc:seculence></pre>
<pre></pre>
twoe-"Library" minOccure-"0"
mexOccurs="unbounded"/>
maxOccurs= unbounded />
trme-"PeliceStation" minOccura-"0"
type= roncestation innoccurs= 0
maxOccurs= unbounded />
<xs.autioute name="name" type="xs.string"></xs.autioute>
<xs:complextype name="Library"></xs:complextype>
<xs.sequence></xs.sequence>
<xs.clellicit liallic="book" type="book</th"></xs.clellicit>
<xs.autibute name="name=" type="xs.string"></xs.autibute>
<xs.complextype name="Dook"></xs.complextype>
<pre><xs:sequence> </xs:sequence></pre>
<pre><xs:element <="" name="author_name" pre=""></xs:element></pre>
tune-"veistring" maxOccurs-"unbounded"/>
/ws:sequences
<vs:complextype< p=""></vs:complextype<>
<pre><vs:securence></vs:securence></pre>
<pre><xs:sequences< pre=""></xs:sequences<></pre>
type="Officers"/>
<pre></pre>
type="xs:string"/>
<pre><xs:complextype name="Officers"></xs:complextype></pre>
<xs:sequence></xs:sequence>
<xs:element <="" name="name" th="" type="xs:string"></xs:element>
minOccurs="1" maxOccurs="unbounded"/>

[0403] A first target XML schema for police stations is given by:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLschema"
elementFormDefault="qualified"
                           attributeFormDefault="unqualified">
<xs:element name="PoliceStations'
type="PoliceStations"/>
<xs:complexType name="PoliceStations">
    <xs:sequence>
         <xs:element name="Station" type="Station"</pre>
        minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="Station">
    <xs:sequence>
         <xs:element name="Officers"</pre>
         type="Officers"/>
```

</ts:sequence> <ss:attribute name="identifier" type="xs:string"/> </ts:complexType name="Officers"> <ss:complexType name="Officers"> <ss:sequence> <ts:sequence> </ts:sequence> </ts:sequence> </ts:sequence> </ts:sequence> </ts:sequence> </ts:sequence>

[0404] A common ontology model for the source and target XML schema is illustrated in **FIG. 26**. A mapping of the source XML schema into the ontology model is given by:

TABLE CXXIX

Mapping from Source schema to Ontology for Twenty-Third Example			
schema	Ontology	Property Index	
complexType: book	Class: Book		
element: book/title/text()	Property: title(Book)	2	
element: book/author	Property: author(Book)	1	
name/text()			
complexType: library	Class: Library		
element: library/books	Container Class:	5	
	set[Book]		
element: library/@name	Property: name(Library)	6	
complexType: officer	Class: Person		
element: officer/name/text()	Property: name(Person)	7	
complexType: police_station	Class: Station		
element: police_station/officers	Container Class:	8	
	set[Person]		
element: police_station/	Property:	9	
@identifier	identifier(Station)		
complexType: town	Class: Town		
element: town/libraries	Container Class:	3	
	set[Library]		
element: town/police_stations	Container Class:	10	
	set[Station]		
element: town/@name	Property: name(Town)	4	

[0405] A mapping of the first target XML schema into the ontology model is given by:

TABLE CXXX

Mapping for	rom Target schema to Ontology Twenty-Third Example	_
schema	Ontology	Property Index
complexType: officer	Class: Person	
element:	Property: name(Person)	7
officer/name/text()		
complexType: station	Class: Station	
element: station/officers	Container Class:	8
	set[Person]	
element:	Property:	9
station/@identifier	identifier(Station)	
complexType:	Class:	
police_stations	set[Station]	

[0406] Based on Tables CXXIX and CXXX, an XSLT transformation that maps XML documents that conform to

the source schema to corresponding documents that conform to the first target schema is given by:

xml version="1.0" encoding="UTF-8"? <xsl:stylesheet <br="" version="1.0">xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <xsl:output <br="" method="xml" version="1.0">encoding="UTF-8" indent="yes"/> <xsl:template match="/"> <policestations></policestations></xsl:template></xsl:output></xsl:stylesheet>
<xsi:ior-each select=".//PoliceStation"></xsi:ior-each>
<station></station>
<xsl:attribute name="identifier"></xsl:attribute>
<xsl:value-of select="@identifier"></xsl:value-of>
<xsl:for-each select="Officers"></xsl:for-each>
<officers></officers>
<xsl:for-each< td=""></xsl:for-each<>
select="name[position() <11]">
<xsl:element name="name"></xsl:element>
<xsl:value-of select="."></xsl:value-of>
/vsl:stylesheet
~/A01.01/001001/

[0407] A second target XML schema for temperature in Centigrade is given by:

xml version="1.0" encoding="UTF-8"?
<xs:schema <="" td="" xmlns:xs="http://www.w3.org/2001/XMLschema"></xs:schema>
elementFormDefault="qualified"
attributeFormDefault="unqualified">
<xs:element <="" name="PoliceStations" td=""></xs:element>
type="PoliceStations"/>
<xs:complextype name="PoliceStations"></xs:complextype>
<xs:sequence></xs:sequence>
<pre><xs:element <="" name="Station" pre="" type="Station"></xs:element></pre>
minOccurs="0" maxOccurs="unbounded"/>
<xs:complextype name="Station"></xs:complextype>
<xs:sequence></xs:sequence>
<xs:element name="Officers" type="Officers"></xs:element>
<xs:attribute name="identifier" type="xs:string"></xs:attribute>
<xs:complextype name="Officers"></xs:complextype>
<xs:sequence></xs:sequence>
<xs:element <="" name="name" td="" type="xs:string"></xs:element>
minOccurs="10" maxOccurs="unbounded"/>

[0408] Based on Tables CXXIX and CXXX, an XSLT transformation that maps XML documents that conform to the source schema to corresponding documents that conform to the second target schema is given by:

<?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/> <xsl:template match="/"> <PoliceStations> <xsl:for-each select=".//PoliceStation"> <Station> <xsl:attribute name="identifier"> <xsl:value-of select="@identifier"/> </xsl:attribute> <xsl:for-each select="Officers"> <Officers> <xsl:for-each select="name"> <xsl:element name="name"> <xsl:value-of select="."/> </xsl:element> </xsl:for-each> </Officers> </xsl:for-each> <xsl:call-template name="generate_officer"> <xsl:with-param name="so_far" select="count(name)"/> </xsl:call-template> </Station> </xsl:for-each> </PoliceStations> </xsl:template> <xsl:template name="generate_officer"> <xsl:param name="so far"/> <xsl:if test="\$so_far < 10"> <bar> </bar> <xsl:call-template name="generate_officer"> <xsl:with-param name="so_far" select="\$so far + 1"/> </xsl:call-template> </xsl:if> </xsl:template> </xsl:stylesheet>

[0409] A third target XML schema for temperature in Centigrade is given by:

<?xml version="1.0" encoding="UTF-8"?> <xs:schema xmlns:xs="http://www.w3.org/2001/XMLschema" elementFormDefault="qualified" attributeFormDefault="unqualified"> <xs:element name="PoliceStations" type="PoliceStations"/> <xs:complexType name="PoliceStations"> <xs:sequence> <xs:element name="Station" type="Station" minOccurs="0" maxOccurs="unbounded"/> </xs:sequence> </xs:complexType> <xs:complexType name="Station"> <xs:sequence> <xs:element name="Officers" type="Officers"/> </xs:sequence> <xs:attribute name="identifier"type="xs:string"/> </xs:complexType> <xs:complexType name="Officers"> <xs:sequence> <xs:element name="name" type="xs:string"minOccurs="10" maxOccurs="20"/> </xs:sequence> </xs:complexType> </xs:schema>

[0410] Based on Tables CXXIX and CXXX, an XSLT transformation that maps XML documents that conform to the source schema to corresponding documents that conform to the first target schema is given by:

<?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <xsl:output method="xml" version="1.0" encoding="UTF-8"indent="yes"/> <xsl:template match="/"> <PoliceStations> <xsl:for-each select=".//PoliceStation"> <Station> <xsl:attribute name="identifier"> <xsl:value-of select="@identifier"/> </xsl:attribute> <xsl:for-each select="Officers"> <Officers> <xsl:for-each select="name[position()< 11]"> <xsl:element name="name"> <xsl:value-of select="."/> </xsl:element> </xsl:for-each> </Officers> </xsl:for-each> <xsl:call-template name="generate_officer"> <xsl:with-param name="so_far' select="count(name)"/> </xsl:call-template> </Station> </xsl:for-each> </PoliceStations> </xsl:template> <xsl:template name="generate_officer"> <xsl:param name="so_far"/> <xsl:if test="\$so_far < 20"> <bar> </bar> <xsl:call-template name="generate_officer"> <xsl:with-param name="so_far' select="\$so far + 1"/> </xsl:call-template> </xsl:if> </xsl:template> </xsl:stylesheet>

[0411] Implementation Details—SOL Generation

[0412] As mentioned hereinabove, and described through the above series of examples, in accordance with a preferred embodiment of the present invention a desired transformation from a source RDBS to a target RDBS is generated by:

- **[0413]** (i) mapping the source and target RDBS into a common ontology model;
- **[0414]** (ii) representing fields of the source and target RDBS in terms of properties of the ontology model, using symbols for properties;
- **[0415]** (iii) deriving expressions for target symbols in terms of source symbols; and
- **[0416]** (iv) converting the expressions into one or more SQL queries.

[0417] Preferably the common ontology model is built by adding classes and properties to an initial ontology model, as required to encompass tables and fields from the source and target RDBS. The addition of classes and properties can be

performed manually by a user, automatically by a computer, or partially automatically by a user and a computer in conjunction.

[0418] Preferably, while the common ontology model is being built, mappings from the source and target RDBS into the ontology model are also built by identifying tables and fields of the source and target RDBS with corresponding classes and properties of the ontology model. Fields are preferably identified as being either simple properties or compositions of properties.

[0419] In a preferred embodiment of the present invention, automatic user guidance is provided when building the common ontology model, in order to accommodate the source and target RDBS mappings. Specifically, while mapping source and target RDBS into the common ontology model, the present invention preferably automatically presents a user with the ability to create classes that corresponds to tables, if such classes are not already defined within the ontology. Similarly, the present invention preferably automatically present a user with the ability to create properties that correspond to fields, if such properties are not already defined within the ontology.

[0420] This automatic guidance feature of the present invention enables users to build a common ontology on the fly, while mapping the source and target RDBS.

[0421] In a preferred embodiment of the present invention, automatic guidance is used to provide a user with a choice of properties to which a given table column may be mapped. Preferably, the choice of properties only includes properties with target types that are compatible with a data type of the given table column. For example, if the given table column has data type VARCHAR2, then the choice of properties only includes properties with target type string. Similarly, if the given table column is a foreign key to a foreign table, then the choice of properties only includes properties whose target is the class corresponding to the foreign table.

[0422] In a preferred embodiment of the present invention, automatic guidance is provided in determining inheritance among classes of the common ontology. Conditions are identified under which the present invention infers that two tables should be mapped to classes that inherit one from another. Such a condition arises when a table, T_1 , contains a primary key that is a foreign key to a table, T_2 . In such a situation, the present invention preferably infers that the class corresponding to T_1 inherits from the class corresponding to T_2 .

[0423] For example, T_1 may be a table for employees with primary key Social_Security_No, which is a foreign key for a table T_2 for citizens. The fact that Social_Security_No serves both as a primary key for T_1 and as a foreign key for T_2 implies that the class Employees inherits from the class Citizens.

[0424] Preferably, when the present invention infers an inheritance relation, the user is given an opportunity to confirm or decline. Alternatively, the user may not be given such an opportunity.

[0425] Preferably, representing fields of the source and target RDBS in terms of properties of the ontology model is performed by identifying a key field among the fields of a table and expressing the other fields in terms of the identified

key field using an inverse property symbol for the key field. For example, if a key field corresponds to a property denoted by 1, and a second field corresponds to a property denoted by 2, then the relation of the second field to the first field is denoted by 201^{-1} . If a table has more than one key field, then preferably symbols are listed for each of the key fields, indicating how the other fields relate thereto. For example, if the second field above also is a key field, then the relation of the first field to the second field is denoted by 102^{-1} , and both of the symbols 201^{-1} and 102^{-1} are listed.

[0426] Preferably, deriving expressions for target symbols in terms of source symbols is implemented by a search over the source symbols for paths that result in the target symbols. For example, if a target symbol is given by 301^{-1} , then chains of composites are formed starting with source symbols of the form $a01^{-1}$, with each successive symbol added to the composite chain inverting the leftmost property in the chain. Thus, a symbol ending with a^{-1} is added to the left of the symbol $a01^{-1}$, and this continues until property 3 appears at the left end of the chain.

[0427] Preferably, converting symbol expressions into SQL queries is accomplished by use of Rules 1-7 described hereinabove with reference to the examples.

[0428] Preferably, when mapping a table to a class, a flag is set that indicates whether it is believed that the table contains all instances of the class.

[0429] Implementation Details—XSLT Generation Algorithm

[0430] 1. Begin with the target schema. Preferably, the first step is to identify a candidate root element. Assume in what follows that one such element has been identified—if there are more than one such candidate, then preferably a user decides which is to be the root of the XSLT transformation. Assume that a <root> element has thus been identified. Create the following XSLT script, to establish that any document produced by the transformation will at minimum conform to the requirement that its opening and closing tags are identified by root:

<xsl:template match="/"> <root> </root> </xsl:template>

[0431] 2. Preferably, the next step is to identify the elements in the target schema that have been mapped to ontological classes. The easiest case, and probably the one encountered most often in practice, is one in which the root itself is mapped to a class, be it a simple class, a container class or a cross-product. If not, then preferably the code-generator goes down a few levels until it comes across elements mapped to classes. The elements that are not mapped to classes should then preferably be placed in the XSLT between the <root> tags mentioned above, in the correct order, up to the places where mappings to classes begin.

TABLE CXXXI-continued

<xsl:template match="/"> <root></root></xsl:template>
<sequence1></sequence1>
<pre>[<element1> mapped to class]</element1></pre>
<element2></element2>
<sequence2></sequence2>

[0432] 3. Henceforth, for purposes of clarity and exposition, the XSLT script generation algorithm is described in terms of an element <fu> that is expected to appear in the target XML document and is mapped to an ontological class, whether that means the root element or a parallel set of elements inside a tree emanating from the root. The treatment is the same in any event from that point onwards.

[0433] 4. Preferably the XSLT generation algorithm divides into different cases depending on a number of conditions, as detailed hereinbelow:

TABLE CXXXI

Conditions for <xsl:for-each> Segments</xsl:for-each>	
Condition	XSLT Segment
- fu> is mapped to a simple class Foo with cardinality parameters minOccurs="1" maxOccurs="1" in the XML schema and there is a corresponding element <foo> in the source document that is associated to the same place Fee</foo>	А
class Foo. (fu> is mapped to a simple class Foo with cardinality parameters minOccurs="0" maxOccurs="1" in the XML schema and there is a corresponding element <foo> in the source document that is associated to the same class Foo.</foo>	В
class root. <fus> is mapped to a container class set[Foo] with cardinality parameters minOccurs="0" maxOccurs="unbounded" in the XML schema, and there are corresponding elements <foos1>, <foos2>,, <foosn> in the source document each of which is associated to the same container-class set[Foo]</foosn></foos2></foos1></fus>	С
thus solver of, fuss is mapped to a container class set[Foo] with cardinality parameters minOccurs="0" maxOccurs="unbounded" in the XML schema, but there is no corresponding element <foos> in the source document that is associated with the same container-class set[Foo]. There are, however, perhaps elements <fool>, <foo2> <foom> which are each individually mapped to the class Foo</foom></foo2></fool></foos>	D
<pre>cfuss is mapped to a container class set[Foo] with cardinality parameters minOccurs="0" maxOccurs="n" in the XML schema, and there are corresponding elements <foos1>, <foos2>,, <foosk> in the source document each of which is associated to the same container class set[Foo]</foosk></foos2></foos1></pre>	Е
<pre>slasterined to the container class set[Foo] with cardinality parameters minOccurs="0" maxOccurs="n" in the XML schema, but there is no corresponding element <foos> in the source document that is associated with the same container- class set[Foo]. There are, however, perhaps elements <fool>, <foo2> <fook> which are each individually mapped to the class Foo.</fook></foo2></fool></foos></pre>	F
fus> is mapped to a container class set[Foo] with cardinality parameters minOccurs="m" maxOccurs="n" in the XML schema, and there	G

	Conditions for <xsl:for-each> Segments</xsl:for-each>	
Condition		XSLT Segment
are corresponding <foosk> in the so associated to the fus> is mapped t cardinality param maxOccurs="n" there is no corres source document container-class so</foosk>	g elements <foos1>, <foos2>,, ource document each of which is same container-class set[Foo]. o a container class set[Foo] with neters minOccurs="m" in the XML schema, but sponding element <foos> in the that is associated with the same et[Foo]. There are, however,</foos></foos2></foos1>	Н
perhaps elements are each individu	<foo1>, <foo2> <fook> which hally mapped to the class Foo.</fook></foo2></foo1>	

[0434] For cases C and D, the XML schema code preferably looks like:

<xsd:complextype name="fus"></xsd:complextype>	
<xsd:sequence></xsd:sequence>	
<pre><xsd:element <="" minoccurs="0" name="fu" pre="" type="fu_view"></xsd:element></pre>	
maxOccurs="unbounded"/>	

[0435] For cases E and F, the XML schema code preferably looks like:

<xsd:complextype name="fus"></xsd:complextype>
<xsd:sequence></xsd:sequence>
<pre><xsd:element <="" minoccurs="0" name="fu" pre="" type="fu_view"></xsd:element></pre>
maxOccurs="n">

[0436] For cases G and H, the XML schema code preferably looks like:

	_
<xsd:complextype name="fus"></xsd:complextype>	
<xsd:sequence></xsd:sequence>	
<pre><xsd:element <="" minoccurs="0" name="fu" pre="" type="fu_view"></xsd:element></pre>	
maxOccurs="n">	

[0437] For the rules as to what should appear in between the <for-each> tags, see step 5 hereinbelow.

CASE A:

<fu>

<xsl:for-each select=".//foo[position() = 1"> </xsl:for-each> </fu> CASE B: <xsl:for-each select=".//foo[position() = 1]"> <fu> </fu> </xsl:for-each> CASE C: <fus> <xsl:for-each select=".//foos1"> <xsl:for-each select="foo"> <fu> </fu> </xsl:for-each> </xsl:for-each> <xsl:for-each select=".//foos2"> <xsl:for-each select="foo"> <fu> </fu> </xsl:for-each> </xsl:for-each> <xsl:for-each select=".//foosn"> <xsl:for-each select="foo"> <fu> </fu> </xsl:for-each> </xsl:for-each> </fus> CASE D: <fus> <xsl:for-each select=".//foo1"> <fu> </fu> </xsl:for-each> <xsl:for-each select=".//foo2"> <fu> </fu> </xsl:for-each> <xsl:for-each select=".//foom"> <fu> </fu> </xsl:for-each> </fus> CASE E: <xsl:template match="/"> <fus> <xsl:call-template name="find foos1"> <xsl:with-param name="so_far" select="0"/> </xsl:call-template> </fus> </xsl:template> <xsl:template name="find_foos1"> <xsl:param name="so_far"/> <xsl:if test="\$so_far < n+1"> <xsl:for-each select=".//foos1/foo"> <xsl:if test="\$so_far+position() < n+1"> <fu> </fu> </xsl:if> </xsl:for-each> </xsl:if> <xsl:call-template name="find_foos2"> <xsl:with-param name="so_far" select="\$so_far+count(.//foos1/foo)"/> </xsl:call-template>

-continued

</xsl:template> <xsl:template name="find_foos2"> <xsl:param name="so_far"/>
<xsl:if test="\$so_far < n+1"> <xsl:for-each select=".//foos2/foo">
 <xsl:fit test="\$so_far+position() < n+1"> <fu> </fu> </xsl:if> </xsl:for-each> </xsl:if> <xsl:call-template name="find_foos3"> <xsl:with-param name="so_far" select="\$so_far+count(.//foos2/foo)"/> </xsl:call-template> </xsl:template> <xsl:template name="find_foosk"> <xsl:for-each select=".//foosn/foo"> <xsl:if test="\$so_far+position() < n+1"> <fu> </fu> </xsl:if> </xsl:for-each> </xsl:if> </xsl:template> CASE F: <xsl:template match="/"> <fus> <xsl:call-template name="find_foo 1 "> <xsl:with-param name="so_far" select="0"/> </xsl:call-template> </fus> </xsl:template> <xsl:template name="find_foo1"> <xsl:param name="so_far"/> <xsl:if test="\$so_far < n+1"> <xsl:for-each selects".//foo1"> <xsl:if test="\$so_far+position() < n+1"> <fu> </fu> </xsl:if> </xsl:for-each> </xsl:if> <xsl:call-template name="find_foo2"> <xsl:with-param name="so_far" select="\$so_far+count(.//foo1)"/> </xsl:call-template> </xsl:template> <xsl:template name="fmd_foo2"> <xsl:param name="so_far"/> <xsl:if test="\$so_far < n+1"> <xsl:for-each select=".//foo2"> <xsl:if test="\$so_far+position() < n+1"> <fu> </fu> </xsl:if> </xsl:for-each> </xsl:if> <xsl:call-template name="find_foo3"> <xsl:with-param name="so_far" select="\$so_far+count(.//foo2)"/> </xsl:call-template> </xsl:template> <xsl:template name="find_fook"> <xsl:param name="so_far"/> <xsl:if test="\$so_far < n+1"> <xsl:for-each select=".//fook"> <xsl:if test="\$so_far+position() < n+1"> <fu> </fu> </xsl:if> </xsl:for-each> </xsl:if </xsl:template>

CASE G:

-continued

```
<xsl:template match="/">
    <fus>
          <xsl:call-template name="find_foos1">
               <xsl:with-param name="so_far" select="0"/>
          </xsl:call-template>
     </fus>
</xsl:template>
<xsl:template name="find foos1">
     <xsl:param name="so_far"/>
          <\!\!\mathrm{xsl:if\ test}=``\$so\_far < n\!+\!1"\!>
              <xsl:for-each select=".//foos1/foo">
                    <xsl:if test="$so_far+position() < n+1">
                        <fu>
                        </fu>
                    </xsl:if>
         </xsl:for-each>
         </xsl:if>
          <xsl:call-template name="findd_foos2">
               <xsl:with-param name="so_far" select="$so_far+count(.//foos1/foo)"/>
          </xsl:call-template>
</xsl:template>
<xsl:template name="find foos2">
     <xsl:param name="so_far"/>
          <\!\!\mathrm{xsl:if\ test=``\$so\_far\ < n+1''\!\!>}
              <xsl:for-each select=".//foos2/foo">
                   <xsl:if test="$so far+position( ) < n+1">
                        <fu>
                        </fu>
                    </xsl:if>
              </xsl:for-each>
         </xsl:if>
          <xsl:call-template name="find_foos3">
              <xsl:with-param name="so_far" select="$so_far+count(.//foos2/foo)"/>
         </xsl:call-template>
</xsl:template>
<xsl:template name="find_foosn">
     <xsl:param name="so_far"/>
          <xsl:if test="$so_far < k+1">
              <xsl:for-each select=".//foosn/foo">
                    <xsl:if test="$so_far+position() < n+1">
                        <fu>
                        </fu>
                    </xsl:if>
              </xsl:for-each>
          </xsl:if>
          <xsl:call-template name="generate_fus">
              <xsl:with-param name="so_far" select="$so_far+count(.//foosk/foo)"/>
         </xsl:call-template>
</xsl:template>
<xsl:templatename="generate_fus">
     <xsl:param name="so_far"/>
     <\!\! xsl:if test="$so_far < m"\!\!>
         <fu>
          </fu>
          <xsl:call-template name="generate_fus">
              <xsl:with-param name="so_far" select="$so_far + 1"/>
          </xsl:call-template>
     </xsl:if>
</xsl:template>
```

44

-continued

CASE H:

<xsl:template match="/"> <fus> <xsl:call-template name="find foo1"> <xsl:with-param name="so_far" select="0"/> </xsl:call-template> </fus> </xsl:template> <xsl:template name="find_foo1"> <xsl:param name="so_far"/> <xsl:if test="\$so_far < n+1"> <xsl:for-each select=".//foo1"> <xsl:if test="\$so_far+position() < n+1"> <fu> </fu> </xsl:if> </xsl:for-each> </xsl:if> <xsl:call-template name="find_foo2 "> <xsl:with-param name="so_far" select="\$so_far+count(.//foo1)"/> </xsl:call-template> </xsl:template> <xsl:template name="find_foo2"> <xsl:param name="so_far"/> <xsl:if test="\$so_far < n+1"> <xsl:for-each select=".//foo2"> <xsl:if test="\$so_far+position() < n+1"> <fu> </fu> </xsl:if> </xsl:for-each> </xsl:if> <xsl:call-template name="find_foo3 "> <xsl:with-param name="so_far" select="\$so_far+count(.//foo2)"/> </xsl:call-template> </xsl:template> <xsl:template name="find_foon"> <xsl:param name="so_far"/> <xsl:if test="\$so_far < k+1"> <xsl:for-each select=".//foon"> <xsl:if test="\$so_far+position() < n+1"> <fu> </fu> </xsl:if> </xsl:for-each> </xsl:if> <xsl:call-template name="generate_fus"> <xsl:with-param name="so_far" select="\$so_far+count(.//fook)"/> </xsl:call-template> </xsl:template> <xsl:template name="generate_fus"> <xsl:param name="so_far"/> $<\!\!xsl:if test="$so_far < m"\!>$ <fu> </fu> <xsl:call-template name="generate_fus"> <xsl:with-param name="so_far" select="\$so_far + 1"/> </xsl:call-template> </xsl:if> </xsl:template>

[0438] 5. Next assume that the classes have been taken care of as detailed hereinabove in step 4. Preferably, from this point onwards the algorithm proceeds by working with properties rather than classes. Again, the algorithm is divided up into cases.

[0439] Assume that the <fu> </fu> tags have been treated, and that the main issue now is dealing with the elements

 bar> that are properties of <fu>.

[0440] Sequence Lists

[0441] Suppose that the properties of $\langle fu \rangle$ are listed in a sequence complex-type in the target schema. Assume, for the sake of definitiveness, that a complexType fu is mapped to an ontological class Foo, with elements bar_i mapped to respective property, Foo.bar_i. Assume further that the source XML schema has an Xpath pattern fill that maps to the ontological class Foo, with further children patterns fu1/barr1, fu1/barr2, etc., mapping to the relevant property paths.

[0442] In a preferred embodiment of the present invention, specific pieces of code are generated to deal with different maximum and minimum occurrences. Such pieces of code are generated inside the $\langle fu \rangle \langle /fu \rangle$ tags that were generatedas described hereinabove. Preferably, the general rule for producing such pieces of code is as follows:

TABLE CXXXI

Conditions for Filling in <xsl:for-each> Segments

Condition	XSLT Segment
The target XML code says <xs:element <br="" name="bar">minOccurs="1" maxOccurs="1"/> or equivalently <xs:element name="bar"></xs:element>, and</xs:element>	Ι
the source has an associated tag <barr>. The target XML code says <xs:element <br="" name="bar">minOccurs="0" maxOccurs="unbounded"/></xs:element></barr>	J
and the source has an associated tag bar>. The XML code says <xs:element <br="" name="bar">minOccurs="0" maxOccurs="n"/></xs:element>	L
and the source has an associated tag barr>. The XML code says <xs:element <br="" name="bar">minOccurs="m" maxOccurs="unbounded"/></xs:element>	М
where m > 0, and the source has an associated tag barr>. The XML code says <xs:element <br="" name="bar"></xs:element> minOccurs="m" maxOccurs="n"/>	Ν
where $m > 0$, and n is a finite integer, and the source has an associated tag sarr>. The target sequence includes a line <xs: </xs: element	О
name="bar' minOccurs="m" maxOccurs="n"/> where $m > 0$, but the source has no associated tag.	

[0443]

CASE I:

<bar>
</bar>
</bar>
CASE J:
</style="text-align: center;"></bar>
CASE J:
</style="text-align: center;"></style="text-align: center;"</style="text-align: center;"></style="text-align: center;"</style="text-align: center;"></style="text-align: center;"></style="text-align: center;"</style="text-align: center;"></style="text-align: center;"</style="text-align: center;"></style="text-align: center;"</style="text-align: center;"></style="text-align: center;"</style="text-align: center;"></style="text-align: center;"</style="text-align: center;"</style="text-align: center;"></style="text-align: cente

</br>

<xsl:for-each select="barr[position() < n+1]"> <bar> </bar> </bar></ksl:value-of select="."/> </bar>

CASE L: <xsl:for-each select="barr">

-continued

<xsl:call-template name="generate_bar"></xsl:call-template>
<xsl:for-each select="barr[position() < n+1]">
<xsl:value-of select="."></xsl:value-of>
<xsl:call-template name="generate_bar"></xsl:call-template>
<xsl:with-param name="so_far" select="count(barr)"></xsl:with-param>
<xsl:template name="generate_bar"></xsl:template>
<xsl:param name="so_far"></xsl:param>
<xsl:if test="\$so_far < m"></xsl:if>
<xsl:call-template name="generate_bar"></xsl:call-template>
<xsl:with-param name="so_far" select="\$so_far + 1"></xsl:with-param>
CASE N:
chars

[0444] As an exemplary illustration, suppose the complex-Type appears in the target schema as follows:

```
<xs:complexType name="fu">

<xs:sequence>

<xs:element name="bar1" type="xs:string" />

<xs:element name="bar2" type="xs:string" minOccurs="0"

maxOccurs="7"/>

<xs:element name="bar3" type="xs:string" minOccurs="1"

maxOccurs="8"/>

<xs:element name="bar4" type="xs:string" minOccurs="3"

maxOccurs="unbounded"/>

<xs:element name="bar5" type="xs:string" minOccurs="0"

maxOccurs="unbounded"/>

<xs:element name="bar5" type="xs:string" minOccurs="0"

maxOccurs="unbounded"/>

<xs:element name="barn" type="xs:string" />

</xs:complexType>
```

[0445] Then, based on the above cases, the following XSLT script is generated.

```
<fu>
     <barr1>
              <xsl:value-of select="bar1"/>
    </barr1>
    <xsl:for-each select="bar2[position() < 5]">
              <barr2>
                   <xsl:value-of select="."/>
              </barr2>
    </xsl:for-each>
    <xsl:for-each select="bar3[position() < 9]">
              <barr3>
                        <xsl:value-of select="."/>
              </barr3>
    </xsl:for-each>
    <xsl:call-template name="generate_barr3">
              <xsl:with-param name="so_far"
              select="count(bar3)"/>
```

-continued

</xsl:call-template> <xsl:for-each select="bar4"> <barr4> <xsl:value-of select="."/> </barr4> </xsl:for-each> <xsl:call-template name="generate_barr4"> <xsl:with-param name="so_far" select="count(bar4)"/> </xsl:call-template> <xsl:for-each select="bar5"> <barr5> <xsl:value-of select="."/> </barr5> </xsl:for-each> </xsl:if> </fu> </xsl:template> <xsl:template match="text()|@*"/> <xsl:template name="generate_barr3"> <xsl:param name="so_far"/> <xsl:if test="\$so_far < 1"> <barr3> </barr3> <xsl:call-template name="generate_barr3 "> <xsl:with-param name="so_far" select="\$so_far + 1"/> </xsl:call-template> </xsl:if> </xsl:template> <xsl:template name="generate_barr4"> <xsl:param name="so far"/> <xsl:if test="\$so_far < 3"> <barr4> </barr4> <xsl:call-template name="generate_barr4"> <xsl:with-param name="so_far" select="\$so_far + 1"/> </xsl:call-template> </xsl:if> </xsl:template>

[0446] Choice Lists

[0447] Suppose that the properties of $\langle fu \rangle$ are listed in a choice complex-type in the target schema. Assume again, as above, that fu is mapped to an ontological class Foo, with each of bar_i mapped to a property, Foo.bar_i. Assume further, as above, that the source XML schema has an Xpath pattern foo that maps to the ontological class Foo, with further children patterns foo/barr1, foo/barr2, etc., mapping to the relevant property paths.

[0448] Preferably, the general rule for producing XSLT script associated with a target choice bloc is as follows. Start with the tags <xsl:choose> </xsl:choose>. For each element in the choice sequence, insert into the choose bloc <xsl:when test="barr"></xsl:when> and within that bloc insert code appropriate to the cardinality restrictions of that element, exactly as above for sequence blocs, including the creation of new templates if needed. Finally, if there are no elements with minOccurs="0" in the choice bloc, select any tag <barr> at random in the choice bloc, and insert into the right before the closing XSLT. </xsl:choose>. <xsl:otherwise><barr></barr></xsl:otherwise>.

[0449] As an exemplary illustration, suppose the complex-Type appears I the target schema as follows:

```
<xs:choice>

<xs:element name="bar1" type="xs:string" />

<xs:element name="bar2" type="xs:string" minOccurs="0"

maxOccurs="7"/>

<xs:element name="bar3" type="xs:string" minOccurs="1"

maxOccurs="8"/>

<xs:element name="bar4" type="xs:string" minOccurs="3"

maxOccurs="unbounded"/>

<xs:element name="bar5" type="xs:string" minOccurs="0"

maxOccurs="unbounded"/>

<xs:element name="bar5" type="xs:string" minOccurs="0"

<xs:element name="bar5" type="xs:string" minOccurs="0"
```

[0450] Then, based on the above cases, the following XSLT script is generated.

<fu> <xsl:choose> <xsl:when test="bar1"> <barr1> <xsl:value-of select="bar1"/> </barr1> </xsl:when> <xsl:when test="bar2"> <xsl:for-each select="bar2[position() < 8]"> <barr2> <xsl:value-of select="."/> </barr2> </xsl:for-each> </xsl:when> <xsl:when test="bar3"> <xsl:for-each select="bar3[position() < 9]"> <barr3> <xsl:value-of select="."/> <barr3> </xsl:for-each> <xsl:call-template name="generate_barr3"> <xsl:with-param name="so_far" select="count(bar3)"/> </xsl:call-template> </xsl:when> <xsl:when test="bar4"> <xsl:for-each select="bar4"> <barr4> <xsl:value-of select="."/> </barr4> </xsl:for-each> <xsl:call-template name="generate barr4"> <xsl:with-param name="so_far" select="count(bar4)"/> </xsl:call-template> </xsl:when> <xsl:when test="bar5"> <xsl:for-each select="bar5"> <barr5> <xsl:value-of select="."/> </barr5> </xsl:for-each> </wsl> <xsl:otherwise> </xsl:otherwise> </xsl:choose> </fu> </xsl:template> <xsl:template match="text()|@*"/> <xsl:template name="generate_barr3 "> <xsl:param name="so_far"/> <xsl:if test="\$so_far < 1"> <barr3> </barr3>

[0456] Then the following XSLT script is generated.

-continued

<xsl:call-template name="generate_barr3"></xsl:call-template>
<xsl:with-param <="" name="so_far" td=""></xsl:with-param>
select="\$so_far + 1"/>
<xsl:template name="generate barr4"></xsl:template>
<xsl:param name="so_far"></xsl:param>
$$
<xsl:call-template name="generate_barr4"></xsl:call-template>
<xsl:with-param <="" name="so_far" td=""></xsl:with-param>
select=" $$so_far + 1"/>$

[0451] All Lists

[0452] Suppose that the properties of <fu> are listed in an all complex-type in the target schema. Assume again, as above, that foo is mapped to an ontological class Foo, with each of bar_i mapped to a property, Foo.bar_i. Assumer further that the source XML schema has an Xpath pattern foo that maps to the ontological class Foo, with further children patterns foo/barr1, foo/barr2, etc., mapping to the relevant property paths.

[0453] In a preferred embodiment of the present invention, a general rule is to test for the presence of each of the source tags associated with the target tags, by way of

<xs< td=""><th>l:if test="foo"></th></xs<>	l:if test="foo">
	<fu></fu>
	<xsl:value-of select="foo"></xsl:value-of>
<th>l:if></th>	l:if>

[0454] Preferably, if any of the elements has minOccurs= "1" then the negative test takes place as well:

<xsl:if test="not (foo)"></xsl:if>	
<fu></fu>	

[0455] As an exemplary illustration, suppose the complex-Type appears I the target schema as follows:

<xs:complexType name="bar">

<xs:all>

<xs:element name="bar2" type="xs:string" minOccurs="0" maxOccurs="1"/> <xs:element name="bar3" type="xs:string" minOccurs="1" maxOccurs="1"/>

</xs:all>

</xs:complexType>

<fu> <xsl:template match="foo"> <xsl:if test="position() = 1"> <xsl:if test="bar1"> <barr1> <xsl:value-of select="bar1"/> </barr1> </xsl:if> <xsl:if test="bar2"> <barr2> <xsl:value-of select="bar2"/> </barr2> </xsl:if> <xsl:if test="not (bar2)"> <barr2> </harr2> </xsl:if></xsl:if></xsl:template>

[0457] 6. In a preferred embodiment of the present invention, when the elements of foo/bar1, foo/bar2, etc. have been processed as above in step 5, everything repeats in a recursive manner for properties that are related to each of the bar_i elements. That is, if the target XML schema has further tags that are children of bar1, bar2, etc., then preferably each of those is treated as properties of the respective target classes of bar1, bar2, and so on, and the above rules apply recursively.

[0458] Additional Considerations

[0459] In reading the above description, persons skilled in the art will realize that there are many apparent variations that can be applied to the methods and systems described. A first variation to which the present invention applies is a setup where source relational database tables reside in more than one database. The present invention preferably operates by using Oracle's cross-database join, if the source databases are Oracle databases. In an alternative embodiment, the present invention can be applied to generate a first SQL query for a first source database, and use the result to generate a second SQL query for a second source database. The two queries taken together can feed a target database.

[0460] 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 method for deriving transformations for transforming data from one data schema to another, comprising:

receiving a source data schema and a target data schema;

mapping the source data schema into an ontology model;

- mapping the target data schema into the ontology model; and
- deriving a transformation for transforming data conforming to the source data schema into data conforming to the target data scheme using the antelegu model
 - the target data schema, using the ontology model.

2. The method of claim 1 further comprising converting at least one of the source data schema and the target schema from an external format to an internal format.

3. The method of claim 1 further comprising receiving the ontology model.

4. The method of claim 3 further comprising converting the ontology model from an external format to an internal format.

5. The method of claim 1 further comprising generating the ontology model.

6. The method of claim 5 further comprising receiving an initial ontology model, wherein said generating generates the ontology model from the initial ontology model.

7. The method of claim 6 further comprising converting the initial ontology model from an external format to an internal format.

8. The method of claim 1 further comprising generating executable program code that transforms data conforming to the source data schema into data conforming to the target data schema.

9. The method of claim 1 wherein the source data schema is a source table schema describing source data tables, wherein the target data schema is a target table schema describing target data tables, and wherein the source table schema and the target table schema each describes at least one table having columns.

10. The method of claim 9 wherein the source table schema is a source relational database schema describing source relational database tables, wherein the target table schema is a target relational database schema describing target relational database tables, and wherein the transformation is an SQL query.

11. The method of claim 10 wherein said mapping a source data schema and said mapping a target data schema each comprise:

- identifying at least one class in the ontology model corresponding to at least one table; and
- identifying at least one property or composition of properties in the ontology model corresponding to at least one table column.

12. The method of claim 11 wherein said deriving comprises:

labeling properties of the ontology model with symbols;

- converting at least one column in the source relational database schema into at least one source symbol;
- converting at least one column in the target relational database schema into at least one target symbol; and
- expressing the at least one target symbol in terms of at least one source symbol.

13. The method of claim 12 wherein said expressing uses expressions involving composition of properties.

14. The method of claim 12 wherein at least one dependency exists among properties in the ontology model, and wherein said deriving further comprises translating the at least one dependency among properties in the ontology model as at least one dependency between target relational database columns and source relational database columns, and wherein said expressing incorporates the at least one dependency between target relational database columns and source relational database columns and source relational database columns.

15. The method of claim 14 wherein said expressing uses expressions involving arithmetic operations.

16. The method of claim 14 wherein said expressing uses expressions involving character string operations.

17. The method of claim 10 further comprising applying the query to at least one source relational database table to populate at least one target relational database table.

18. The method of claim 17 wherein the at least one source relational database table reside in a single database.

19. The method of claim 17 wherein the at least one source relational database table reside in multiple databases.

20. The method of claim 1 wherein the source data schema is a source document schema describing source documents, and wherein the target data schema is a target document schema describing target documents.

21. The method of claim 20 wherein the source document schema is a source DTD describing source XML documents, wherein the target document schema is a target DTD describing target XML documents, and wherein the source DTD and the target DTD each describes at least one XML element or XML attribute.

22. The method of claim 21 wherein the transformation is an XQuery.

23. The method of claim 21 wherein the transformation is an XSLT script.

24. The method of claim 20 wherein the source document schema is a source XML schema describing source XML documents, wherein the target document schema is a target XML schema describing target XML documents, and wherein the source XML schema and the target XML schema each describes at least one XML complexType having at least one XML element or XML attribute.

25. The method of claim 24 wherein the transformation is an XQuery.

26. The method of claim 24 wherein the transformation is an XSLT script.

27. The method of claim 24 wherein said mapping a source data schema and said mapping a target data schema each comprise:

- identifying at least one class in the ontology model corresponding to at least one XML complexType; and
- identifying at least one property or composition of properties in the ontology model corresponding to at least one XML element or XML attribute.

28. The method of claim 24 wherein said deriving comprises expressing XML elements and XML attributes of the target XML schema in terms of XML elements and XML attributes of the source XML schema.

29. The method of claim 28 wherein said expressing is performed recursively through XPath paths.

30. The method of claim 27 wherein at least one dependency exists among properties in the ontology model, and

wherein said deriving further comprises translating the at least one dependency among properties in the ontology model as at least one dependency between target XML elements and source XML elements.

31. The method of claim 26 further comprising applying the XSLT script to at least one source XML document to generate at least one target XML document.

32. The method of claim 31 wherein the at least one source XML document reside in a single database.

33. The method of claim 31 wherein the at least one source XML document reside in multiple databases.

34. A system for deriving transformations for transforming data from one data schema to another, comprising:

- a schema receiver receiving a source data schema and a target data schema;
- a mapping processor mapping a data schema into an ontology model; and
- a transformation processor deriving a transformation for transforming data conforming to the source data schema into data conforming to the target data schema, based on respective source and target mappings generated by said mapping processor for mapping said source data schema and said target data schema into a common ontology model.

35. The system of claim 34 further comprising a schema format convertor, converting at least one of the source data schema and the target data schema from an external format to an internal format.

36. The system of claim 34 further comprising an ontology receiver receiving the ontology model.

37. The system of claim 36 further comprising an ontology format convertor, converting the ontology model from an external format to an internal format.

38. The system of claim 34 further comprising an ontology builder generating the ontology model.

39. The system of claim 38 further comprising an ontology receiver receiving an initial ontology model, wherein said ontology builder generates the ontology model from the initial ontology model.

40. The system of claim 39 further comprising an ontology format convertor, converting the initial ontology model from an external format to an internal format.

41. The system of claim 34 further comprising a program code generator generating executable program code that transforms data conforming to the source data schema into data conforming to the target data schema.

42. The system of claim 34 wherein the source data schema is a source table schema describing source data tables, wherein the target data schema is a target table schema describing target data tables, and wherein the source table schema and the target table schema each describes at least one data table having columns.

43. The system of claim 42 wherein the source table schema is a source relational database schema describing source relational database tables, wherein the target table schema is a target relational database schema describing target database tables, and wherein the transformation is an SQL query.

44. The system of claim 43 wherein said mapping processor comprises:

- a class identifier identifying at least one class in the common ontology model corresponding to at least one table; and
- a property identifier identifying at least one property or composition of properties in the common ontology model corresponding to at least one table column.

45. The system of claim 44 wherein said property identifier presents a user with a choice of at least one property in the common ontology model that may correspond to a given table column.

46. The system of claim 45 wherein the choice of at least one property only includes properties having targets that are compatible with a data type of the given table column.

47. The system of claim 46 wherein, for a given table column that is a foreign key to a foreign table, the choice of at least one property only includes properties whose target is a class corresponding to the foreign table.

48. The system of claim 43 wherein said transformation processor comprises:

- an ontology labeller labeling properties of the common ontology model with symbols;
- a column converter converting at least one column in the source relational database schema into at least one source symbol, and converting at least one column in the target relational database schema into at least one target symbol; and
- a symbol processor expressing the at least one target symbol in terms of at least one source symbol.

49. The system of claim 48 wherein said symbol processor uses expressions involving composition of properties.

50. The system of claim 48 wherein at least one dependency exists among properties in the ontology model, and wherein said transformation processor further comprises a dependency processor translating the at least one dependency among properties in the ontology model as at least one dependency between target relational database columns and source relational database columns, and wherein said symbol processor incorporates the at least one dependency between target relational database columns and source relational database columns.

51. The system of claim 50 wherein said symbol processor uses expressions involving arithmetic operations.

52. The system of claim 50 wherein said symbol processor uses expressions involving character string operations.53. The system of claim 43 further comprising:

- 55. The system of claim 45 further comprising.
- a data receiver receiving at least one source relational database table; and
- a data processor applying the query to the at least one source relational database table to populate at least one target relational database table.

54. The system of claim 53 wherein the at least one source relational database table reside in a single database.

55. The system of claim 53 wherein the at least one source relational database table resides in multiple databases.

56. The system of claim 34 wherein the source data schema comprises a source document schema describing source documents, and wherein the target data schema comprises a target document schema describing target documents.

57. The system of claim 56 wherein the source document schema is a source DTD describing source XML documents, wherein the target document schema is a target DTD describing target XML documents, and wherein the source DTD and the target DTD each describes at least one XML element or XML attribute.

58. The system of claim 57 wherein the transformation is an XQuery.

59. The system of claim 57 wherein the transformation is an XSLT script.

60. The system of claim 56 wherein the source document schema comprises a source XML schema that describes XML source documents, wherein the target document schema comprises a target XML schema that describes XML target documents, and wherein the source XML schema and the target XML schema each comprises at least one XML complexType having at least one XML element or XML attribute.

61. The system of claim 60 wherein the transformation is an XQuery.

62. The system of claim 60 wherein the transformation is an XSLT script.

63. The system of claim 60 wherein said mapping processor comprises:

- a class identifier identifying at least one class in the ontology model corresponding to at least one XML complexType; and
- an property identifier identifying at least one property or composition of properties in the ontology model corresponding to at least one XML element or XML attribute.

64. The system of claim 60 wherein said transformation processor comprises an XML processor expressing XML elements and XML attributes of said target XML schema in terms of XML elements and XML attributes of said source XML schema.

65. The system of claim 64 wherein said XML processor operates recursively through XPath paths.

66. The system of claim 64 wherein at least one dependency exists among properties in the ontology model, and wherein said transformation processor further comprises a dependency processor translating the at least one dependency among properties in the ontology model as at least one dependency between target XML elements or attributes, and source XML elements or attributes, and wherein said XML processor incorporates the at least one dependency between target XML elements or attributes, and source XML elements or attributes, and source XML elements or attributes, and source XML elements or attributes.

67. The system of claim 60 further comprising

- a data receiver receiving at least one source XML document; and
- a data processor applying the XSLT script to the at least one source XML document to generate at least one target XML document.

68. The system of claim 67 wherein the at least one source XML document reside in a single database.

69. The system of claim 67 wherein the at least one source XML document reside in multiple databases.

70. A method for building an ontology model into which data schema can be embedded, comprising:

receiving at least one data schema; and

building an ontology model into which the at least one data schema can be embedded.

71. The method of claim 70 further comprising converting at least one of the at least one data schema from an external format to an internal format.

72. The method of claim 70 wherein the at least one data schema is at least one table schema describing data tables having columns.

73. The method of claim 72 wherein the at least one table schema is at least one relational database schema describing relational database tables.

74. The method of claim **73** wherein said building an ontology model comprises:

providing an initial ontology model;

- adding classes to the initial ontology model corresponding to tables described in the at least one relational database schema; and
- adding properties to the initial ontology model corresponding to columns described in the at least one relational database schema.

75. The method of claim 74 wherein the initial ontology model is empty.

76. The method of claim 74 wherein the initial ontology model is non-empty.

77. The method of claim 76 further comprising converting the initial ontology model from an external format to an internal format.

78. The method of claim 74 wherein said adding classes is performed by a computer in conjunction with a user.

79. The method of claim 78 wherein said adding classes prompts a user to add a class to the ontology model when there is a table described in the at least one relational database schema that does not correspond to an existing class in the ontology model.

80. The method of claim 74 wherein said adding classes is performed automatically by a computer.

81. The method of claim 80 wherein said adding classes automatically adds a class to the ontology model when there is a table described in the at least one relational database schema that does not correspond to an existing class in the ontology model.

82. The method of claim 74 wherein said adding properties is performed by a computer in conjunction with a user.

83. The method of claim 82 wherein said adding properties prompts a user to add a property to the ontology model when there is a table column described in the at least one relational database schema that does not correspond to an existing property or composition of properties in the ontology model.

84. The method of claim 74 wherein said adding properties is performed automatically by a computer.

85. The method of claim 84 wherein said adding properties automatically adds a property to the ontology model when there is a table column described in the at least one relational database schema that does not correspond to an existing property or composition of properties in the ontology model.

86. The method of claim 70 wherein said building an ontology model comprises inferring inheritance relationships between classes in the ontology model based on relationships between tables described in the at least one relational database schema.

87. The method of claim 86 wherein a first class in the ontology model is inferred to inherit from a second class in the ontology model when a table corresponding to the first class has a primary key that is a foreign key to a table corresponding to the second class.

88. The method of claim 86 wherein said inferring inheritance relationships includes prompting a user to confirm an inferred inheritance relationship.

89. The method of claim 70 wherein the at least one data schema is at least one document schema describing documents.

90. The method of claim 89 wherein the at least one document schema is an XML schema describing XML documents having at least one XML complexType with at least one XML element or XML attribute.

91. The method of claim 90 wherein said building an ontology model comprises:

providing an initial ontology model;

- adding classes to the initial ontology model corresponding to XML complexTypes described in the at least one XML schema; and
- adding properties to the initial ontology model corresponding to XML elements and XML attributes described in the at least one XML schema.

92. The method of claim 91 wherein the initial ontology model is empty.

93. The method of claim 92 wherein the initial ontology model is non-empty.

94. The method of claim 91 wherein said adding classes is performed by a computer in conjunction with a user.

95. The method of claim 94 wherein said adding classes prompts a user to add a class to the ontology model when there is an XML complexType described in the at least one XML schema that does not correspond to an existing class in the ontology model.

96. The method of claim 91 wherein said adding classes is performed-automatically by a computer.

97. The method of claim 96 wherein said adding classes automatically adds a class to the ontology model when there is an XML complexType described in the at least one XML schema that does not correspond to an existing class in the ontology model.

98. The method of claim 91 wherein said adding properties is performed by a computer in conjunction with a user.

99. The method of claim 98 wherein said adding properties prompts a user to add a property to the ontology model when there is an XML element or an XML attribute described in the at least one XML schema that does not correspond to an existing property or composition of properties in the ontology model.

100. The method of claim 91 wherein said adding properties is performed automatically by a computer.

101. The method of claim 100 wherein said adding properties automatically adds a property to the ontology model when there is an XML element or an XML attribute described in the at least one relational database schema that does not correspond to an existing property or composition of properties in the ontology model.

102. A system for building an ontology model into which data schema can be embedded, comprising:

- a schema receiver receiving at least one data schema; and
- a model builder building an ontology model into which the at least one data schema can be embedded.

103. The system of claim 102 further comprising a schema format convertor, converting at least one of the at least one data schema from an external format to an internal format.

104. The system of claim 102 wherein the at least one data schema is at least one table schema describing data tables having columns.

105. The system of claim 104 wherein the at least one table schema is at least one relational database schema describing relational database tables.

106. The system of claim 105 further comprising an ontology receiver receiving an initial ontology model, and wherein said model builder comprises:

- a class adder adding classes to the initial ontology model corresponding to tables described in the at least one relational database schema; and
- a property adder adding properties to the initial ontology model corresponding to table columns described in the at least one relational database schema.

107. The system of claim 106 wherein the initial ontology model is empty.

108. The system of claim 106 wherein the initial ontology model is non-empty.

109. The system of claim 108 further comprising an ontology format convertor, converting the initial ontology model from an external format to an internal format.

110. The system of claim 106 wherein said class adder is guided by a user in conjunction with a computer.

111. The system of claim 110 wherein said class adder prompts a user to add a class to the ontology model when there is a table described in the at least one relational database schema that does not correspond to an existing class in the ontology model.

112. The system of claim 106 wherein said class adder is automatically guided by a computer.

113. The system of claim 112 wherein said class adder automatically adds a class to the ontology model when there is a table described in the at least one relational database schema that does not correspond to an existing class in the ontology model.

114. The system of claim 106 wherein said property adder is guided by a user in conjunction with a computer.

115. The system of claim 114 wherein said property adder prompts a user to add a property to the ontology model when there is a table column described in the at least one relational database schema that does not correspond to an existing property or composition of properties in the ontology model.

116. The system of claim 106 wherein said property adder is automatically guided by a computer.

117. The system of claim 116 wherein said property adder automatically adds a property to the ontology model when there is a table column described in the at least one relational database schema that does not correspond to an existing property or composition of properties in the ontology model.

118. The system of claim 105 wherein said model builder comprises an inheritance processor inferring inheritance relationships between classes in the ontology model based on relationships between tables in the at least one relational database schema.

119. The system of claim 118 wherein said inheritance processor infers that a first class in the ontology model

inherits from a second class in the ontology model when a table corresponding to the first class has a primary key that is a foreign key to a table corresponding to the second class.

120. The system of claim 118 wherein said model builder ensures that classes corresponding to tables in the at least one relational database schema obey the inferred inheritance relationships.

121. The system of claim 120 wherein said inheritance processor prompts a user to confirm an inferred inheritance relationship.

122. The system of claim 102 wherein the at least one data schema comprises at least one document schema describing documents.

123. The system of claim 122 wherein the at least one document schema comprises at least one XML schema that describes XML documents, wherein having at least one XML complexType with at least one XML element or XML attribute.

124. The system of claim 123 further comprising an ontology receiver receiving an initial ontology model, and wherein said model builder comprises:

- a class adder adding classes to the initial ontology model corresponding to XML complexTypes described in the at least one XML schema; and
- a property adder adding properties to the initial ontology model corresponding to table columns in the at least one relational database schema.

125. The system of claim 124 wherein the initial ontology model is empty.

126. The system of claim 124 wherein the initial ontology model is non-empty.

127. The system of claim 124 wherein said class adder is guided by a user in conjunction with a computer.

128. The system of claim 127 wherein said class adder prompts a user to add a class to the ontology model when there is an XML complexType described in the at least one XML schema that does not correspond to an existing class in the ontology model.

129. The system of claim 124 wherein said class adder is automatically guided by a computer.

130. The system of claim 129 wherein said class adder automatically adds a class to the ontology model when there is an XML complexType described in the at least one XML schema that does not correspond to an existing class in the ontology model.

131. The system of claim 124 wherein said property adder is guided by a user in conjunction with a computer.

132. The system of claim 131 wherein said property adder prompts a user to add a property to the ontology model when

there is an XML element or XML attribute described in the at least one XML schema that does not correspond to an existing property or composition of properties in the ontology model.

133. The system of claim 124 wherein said property adder is automatically guided by a computer.

134. The system of claim 133 wherein said property adder automatically adds a property to the ontology model when there is an XML element or XML attribute described in the at least one XML schema that does not correspond to an existing property or composition of properties in the ontology model.

135. An article of manufacture including one or more computer-readable media that embody a program of instructions for transforming data from one schema to another, wherein the program of instructions, when executed by a processing system, causes the processing system to:

receive a source data schema and a target data schema;

map the source data schema into an ontology model;

map the target data schema into the ontology model; and

derive a transformation for transforming data conforming to the source data schema into data conforming to the target relational database schema, using the ontology model.

136. The article of claim 135 wherein the one or more computer-readable media include one or more non-volatile storage devices.

137. The article of claim 135 wherein the one or more compute-readable media include a carrier wave modulated with a data signal.

138. An article of manufacture including one or more computer-readable media that embody a program of instructions for building a common ontology model into which data schema can be embedded, wherein the program of instructions, when executed by a processing system, causes the processing system to:

receive at least one data schema; and

build an ontology model into which the at least one data schema can be embedded.

139. The article of claim 138 wherein the one or more computer-readable media include one or more non-volatile storage devices.

140. The article of claim 138 wherein the one or more compute-readable media include a carrier wave modulated with a data signal.

* * * * *