(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
8 August 2013 (08.08.2013)

WIPO | PCT

(10) International Publication Number
WO 2013/115985 A2

(51) International Patent Classification:
G06F 17/30 (2006.01)

(21) International Application Number:
PCT/US20 13/02 1626

(22) International Filing Date:
16 January 2013 (16.01 .2013)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
61/593,372 1 February 2012 (01.02.2012) US

(71) Applicant *(for all designated States except US):* **SIEMENS CORPORATION** [US/US]; 170 Wood Avenue South, Iselin, New Jersey 08830 (US).

(72) Inventors; and
(71) Applicants *(for US only):* **FRADKIN, Dmitriy** [US/US]; 449 Sayre Drive, Princeton, New Jersey 08540 (US). **MOERCHEN, Fabian** [US/US]; 10967 103rd Avenue SW, Vashon, Washington 98070 (US). **TECUCI, Dan G.** [RO/US]; 3809 Quail Ridge Drive, Plainsboro, New Jersey 08536 (US).

(74) Agents: **CONOVER, Michele L.** et al; Siemens Corporation- Intellectual Property Dept., 170 Wood Avenue South, Iselin, New Jersey 08830 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available):* AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available):* ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *without international search report and to be republished upon receipt of that report (Rule 48.2(g))*

(54) Title: ARCHITECTURE FOR NATURAL LANGUAGE QUERYING IN SERVICE ANALYTICS DOMAINS

(57) Abstract: A natural language interface is used to interact with data in a service analytics database. Domain entities from the database are initially mapped to an entity relationship model including machines, events, sensor values, service notifications, and spare parts records. A natural language query is then parsed, mapped to elements of the entity relation model and translated into a standard query language.

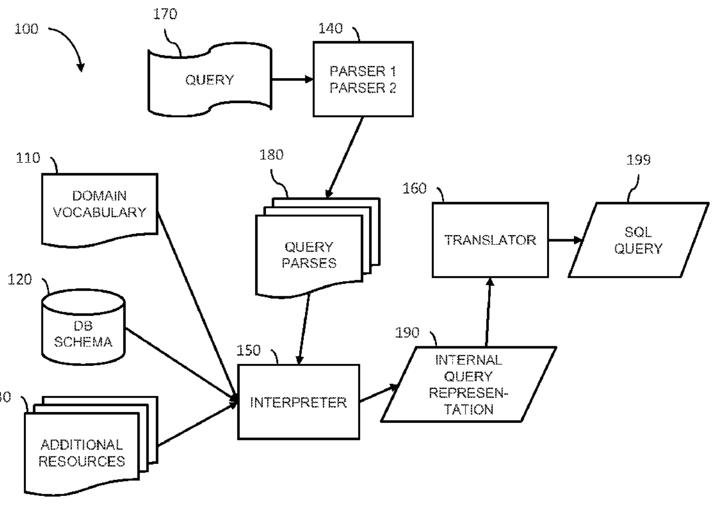

Fig. 1

WO 2013/115985 A2

ARCHITECTURE FOR NATURAL LANGUAGE QUERYING IN SERVICE ANALYTICS DOMAINS

Claim of Priority

[1] This application claims priority to, and incorporates by reference herein in its entirety, pending United States Provisional Patent Application Serial Number 61/593,372, filed February 1, 2012, and entitled "Architecture for Natural Language Querying in Service Analytics Domains."

Field of the Invention

[2] This invention relates generally to techniques for querying databases, and more particularly to methods, systems and computer readable media for natural language querying of databases with a limited domain of business analytics data generated by complex machines or IT systems.

Background of the Invention

[3] The users and maintainers of complex machines and equipment typically collect maintenance data produced during the use and maintenance of that equipment. Those activities generate enormous amounts of information. One example is the use and maintenance of medical imaging and diagnostic equipment. A computer system running a medical device produces log messages as well as databases of measurements of currents, voltages, image quality, diagnostic test results, etc. That information is stored locally on a machine, and is often uploaded to central data repositories for storage and analysis. Additionally, records are made of repair and maintenance activities, as well as

of part replacements with regard to each scanner. Similar situations exist in other technology sectors: in energy (power plants, gas and wind turbines), in cities and infrastructure (trains, smart buildings, water treatment facilities) and in industry (automation and drives). Analysis of such data can provide tremendous benefits, allowing users to gain insights into usage and failure patterns, to identify areas for improved service and usability, and to generate reports and summaries.

[4] Domain experts are best suited to analyze this data. Domain experts are people whose daily work focuses on developing and improving specific equipment under observation, and who are familiar with how these products should properly be used and maintained. A significant obstacle to the access of the relevant data by domain experts is that the data is stored in relational databases, accessible only via SQL queries or custom applications. Most domain experts are not computer scientists and have limited familiarity with IT and databases, and thus are unable to make use of the data available to them. To a certain degree this can be addressed with customizable reporting and dashboard solutions, but those require a significant development effort to build and update, and typically require training to use.

Summary of the Invention

[5] The present invention addresses the needs described above by providing a method for querying a service analytics database. The method includes mapping domain entities from the database to an entity relation model having elements including: machines representing single pieces of equipment; at least one of time-stamped events relating to particular machines and sensor values relating to particular times; and service notifications relating to particular machines and including descriptions of services performed. The elements may also include spare parts records relating to particular machines and relating to the service notifications.

[6] The method further comprises receiving a query containing natural language elements; parsing the query to extract concepts; mapping the concepts to the elements of the entity relation model to create an interpretation of the concepts; and translating the query to a standard query language using the interpretation of the concepts.

[7] In another aspect of the invention, a non-transitory computer-usable medium is provided having computer readable instructions stored thereon for execution by a processor to perform methods for querying a service analytics database as described above.

Brief Description of the Drawings

[8] FIG. 1 is a schematic representation of architectural components used in embodiments of the invention.

[9] FIG. 2 is a schematic representation of an entity relation model for service analytics in accordance with one embodiment of the invention.

[10] FIG. 3 is a chart showing workflow of a natural language query processing in accordance with one embodiment of the invention.

[11] FIG. 4 is a component diagram showing components of the natural language querying application in accordance with one embodiment of the invention.

[12] FIG. 5 is a flow chart showing a method in accordance with one embodiment of the invention.

[13] FIG. 6 is a schematic diagram showing a system in accordance with one embodiment of the invention.

Description of the Invention

[14] The presently described technique is an approach that enables domain experts to ask a broad range of questions about the data using natural language that the experts are comfortable with. The present disclosure does not attempt to address the general and difficult problem of language understanding, or even natural language querying in a general sense. Rather, the focus is on the limited domain of business analytics of data generated by complex machines or IT systems, such as service analytics or condition monitoring. That domain, however, is sufficiently broad to include many different business entity types and situations within the business world. The present disclosure therefore focuses on processing questions as they would typically be encountered in that domain, but with a general architecture that does not require the full list of exact questions beforehand.

[15] Described herein is a flexible architecture for natural language querying of databases in service analytics domains. The architecture comprises multiple modules, some of which may be interchanged or combined together for better performance. The system generates one or more possible interpretations/candidates for the query and its parts, and then combines multiple sources of information, such as high-level ontology, database schema, domain vocabulary, and look-up tables of common values, to resolve ambiguities and generate a Structured Query Language (SQL) query interpretable by the database. The approach relies on a common ontology of service analytics as described below, as well as domain-specific information such as database schema, domain vocabulary, common values, etc.

[16] An overall architecture 100 of the presently described system is shown in FIG. 1. Several data modules are prepared offline during system development. Those modules include at least a domain vocabulary 110 and database schema 120, and may include additional resources 130. Several components perform the query processing, including parsers 140, an interpreter 150 and a translator 160. For each query 170, one or more query parses 180 are generated by the parsers 140. Those parses 180 are combined with additional information from the modules 110, 120 130 and are disambiguated by the interpreter 150. The resulting internal query representation 190 is then converted by the translator 160 to an SQL query 199.

[17] A natural language interface permits a user to interact with data more naturally and flexibly. Accurately interpreting natural language, however, is a non-trivial task. The present disclosure focuses on the narrow topic of queries concerning service analytics data, as stored in a specific database.

[18] In order to translate a natural language query into an appropriate machine query in a query language such as SQL or SPARQL Protocol RDF Query Language (SPARQL), the following tasks must be performed:

1. Represent the domain:
    a. at high level (abstract entities and relations);
    b. at domain-specific level (properties and values of the domain); and

   c. at the database level - database structure, tables, column names.

2. Understand the question:

   a. natural language parsing;

   b. handling of synonyms; and

   c. handling of domain-specific vocabulary.

3. Connect query words to entities and relations.

4. Translate the internal representation of the query to SQL, possibly via some intermediate representations.

[19] Note that the first task (representing the domain) must be accomplished once, at system development time, before any queries are formulated. The remaining tasks, however, must be performed for each query. The above tasks will now be discussed in more detail.

[20] Representing the Domain

[21] While on the surface, medical scanners, power plants, gas turbines and trains appear to have little in common, much of the data involved with operation and maintenance of those systems has basically the same structure. An example domain representation or entity relation model 200 based on data from the medical domain is shown in FIG. 2. The proposed framework requires the domain representation to be defined at system development time.

[22] Referring to FIG. 2, each of the primary entities will be discussed in turn together with several properties. The primary entities are represented in FIG. 2 as ellipses having a solid outline. The remaining properties and relations have dashed outlines.

[23] A machine 210 refers to a single piece of equipment such as a scanner, a train, etc. It is identified by a unique machine code 211. A machine 210 can also have other characteristics, such as type/model 212 and location 213. Those characteristics are domain specific.

[24] An event 220 is a single message produced by software running the subject machine. Each event 220 has an event code 221 which identifies the type of log message, a timestamp 223 and event text 222, which is the content of the message. An event is related to the machine on which it occurs. That relationship can be encoded via a relation, or by specifying a machine code 211 for an event. Events usually have additional properties such as severity 224, subsystem 225, etc. These properties are domain specific.

[25] A sensor value 230 is a numeric analogue of an event 220. It represents a value measured by a particular sensor on a machine at a particular time. Accordingly, a sensor value 230 has properties sensor value 231, sensor type 232 and timestamp 223. The relation to a specific machine is specified similarly to how it is done for events, via a machine code 211. The inventors have found that certain domains may be accurately represented using only one entity selected from the sensor value entity 230 and the event entity 220.

[26] Service notifications 240 represent records of maintenance or repair performed on a particular machine. It is uniquely identified by notification number 241, and has a date 242 associated with it. Service notifications 240 are related to a particular machine via machine code 211. A service notification 240 also has a property notification text 243 that contains a description of the service performed. Additional properties may exist.

[27] A spare part record 250 is a record of a part replacement. The spare part entity 250 is used in representing domains in which part replacement is monitored, and is not necessarily used in all entity models. The part is identified by a spare part number 251. The replacement is always associated with a specific service notification 240, and therefore with a specific machine 210, via notification number 241 and machine code 211. Additional properties, such as spare part name or quantity 252, may exist.

[28] The experience of the inventors with multiple domains and products, such as medical scanners, gas turbines and trains, has shown that data from many domains largely match the schema 200 shown in FIG. 2. It is therefore possible to construct a generic ontology representing these relations, with appropriate vocabulary. For a specific application, however, it is necessary to supplement the generic ontology with appropriate properties and relations, as well as to extend the vocabulary. The various elements of the ontology must furthermore be mapped to the actual database structure. Unfortunately, these tasks cannot be completely automated, but a number of tools exists that can assist with it.

[29] Many tools, such as the open source D2RQ platform, are available that can extract a database schema and represent it as an ontology or as a mapping. WordNet, a lexical database maintained at the Cognitive Science Laboratory of Princeton University, can be used to semi-automatically determine appropriate synonyms to terms of interest or to match words to existing concepts at execution time. Another approach would maintain a mapping file from various words, phrases and contexts to entities and relations of the database.

[30] Additional improvements in matching may be obtained by storing sets of values of some entity attributes in a look-up table. It is then possible to automatically determine relevant entities and relations when queries mention these values.

[31] Understanding and Translating the Question

[32] The first decision to be made when developing a system for answering questions is to determine what kinds of questions will be accepted. The options differ in the flexibility they provide in formulating the query, and also in the ease of developing the rest of the system. Some examples of alternative approaches include keywords and phrases, controlled English and full English. Each of those alternative question formulations is discussed in turn below.

[33] Keywords and Phrases: If a query is formulated as an unstructured set of keywords or short phrases, those keywords and short phrases can be individually mapped to the domain. Parsers can also be applied to phrases, to identify simple relations.

[34] Controlled English: If the query uses a controlled English language such that defined by the Attempto Controlled English (ACE) project of the University of Zurich, in which an unambiguous subset of real English is used, then the relations can be identified unambiguously. The advantage of that approach is that the user can use relatively more expressive queries, such as a sentence containing a subject and object. The query, however, must to conform to the rules prescribed for Controlled English (for example, pronouns cannot be used).

[35] Full English: For an unconstrained question, syntactic parsers and semantic role labelers can be used. Parsers construct syntactic parse trees of the input, and branches can then be compared with the vocabulary to identify items and relations. Semantic role labelers help the system to not only identify the key items or entities, but also to define how they relate to the main proposition expressed in the user question. While this approach gives the user full freedom, it is error prone because language understanding is limited to the state of the art parsers and semantic role labelers, and because the user has more leeway to create ungrammatical sentences that will trip the system.

[36] It is also possible to use an 'ensemble' approach, where multiple question understanding modules are used in parallel. While some modules may fail, others may produce one or more parses, with confidence scores. The system then tries different parses in order of the confidence scores until a database query is successfully performed. Alternatively, the system identifies and combines the elements with highest confidence scores across different translations.

[37] The question or query must be "understood" by the system. This means that in addition to being parsed, the concepts/entities/properties and their relationships must be identified and mapped to the domain-specific database structure. Once the relationships are identified and mapped, the question or query is converted to an internal representation: a set of objects and properties and relations between them.

[38] It remains to formulate the question or query in a way acceptable to the database. Once again, there is no single best solution. Several options may be integrated with the presently described system. The query may be formulated using a query language able to retrieve and manipulate data stored in Resource Description Framework (RDF) format, such as SPARQL. A tool such as D2RS may then be used to access the database and get back the results.

[39] Several JAVA-based formulations are available. The query may be formulated using Criteria API, a JAVA application programming interface for querying databases. The query may be formulated using Java Persistence Query Language (JPQL), a platform-independent object-oriented query language defined as part of the Java Persistence API (JPA) specification, and then possibly translated to Structured Query Language (SQL), and ANSI standard language for querying databases.

[40] Because JPQL currently does not support particular functions and the current hibernate version does not support user defined functions, additional steps are necessary. The functions that are not supported by JPQL are added to the translated SQL query. In that way, the JPQL query is first converted to SQL. The SQL query is then extended and is run by the query engine.

[41] The query may alternatively be formulated in SQL directly. Once the query is formulated in or translated to SQL, it is straightforward to obtain and present the results.

[42] The chart 300, shown in FIG. 3, summarizes each stage of the question processing. Three different query types are illustrated: a complete English sentence 301, a controlled English query 302 and keywords and phrase segments 303. The chart shows alternative approaches for query understanding and translation.

[43] As noted above, query formulation, represented by row 310 of the chart 300, may be done using a variety of formats, depending on the ease of use and the degree of control provided to the user. In the case 301 where the user is permitted to enter any complete English sentence, the user is afforded the greatest ease of use, but has the least control over the question processing. Parsing 320 of a complete English sentence is performed by semantic parsers that attempt to extract relations between words. The parsers are preferably chosen based on the query type, as are the parsers 320 shown in chart 300. The query is then interpreted 330 by identifying the involved entries and relationships, using the domain-specific representation created for the subject database. The interpreted query is then translated 340 using any of the techniques described above, including directly formulating an SQL query, formulating a query via Java Criteria API, or formulating a query in SPARQL, using D2RQ software to produce an SQL query.

[44] Where a controlled English query formulation 302 is used, the user has greater control over question processing, but must utilize a formally defined, unambiguous subset of the English language such as the ACE language. An ACE parser is used in parsing such queries. The controlled English query is converted by the parser into first order logic or an equivalent formulation. Interpretation and translation of such a parsed query may be performed by any of the techniques discussed above.

[45] The query may also be formulated as phrase segments and/or keywords 303. In that case, the query is parsed using an evidence-based analysis that extracts multiple pieces of evidence indicating the types of questions and entities involved.

[46] Components and Communication

[47] The components of the NLQP application according to one exemplary embodiment of the invention, shown in FIG. 4, are responsible for interpreting the parsing result and creating a query from it. Components shown in FIG. 4 with a name starting with "DRS" are specific for processing of Attempto parser results in DRS format. Other components provide services for mapping and the query definition and can be used independently of the used parser. The arrangement shown in FIG. 4 is only one exemplary arrangement used in processing results produced by an Attempto parser; other components and communications between components may be used without departing from the scope of the invention. The components and the communications between them are described in more detail below.

[48] The DRSTranslatorToQuery class 410 provides a method readDRS 411 and is the interface outwards. The method expects an Attempto parser result in DRS format and handles the whole workflow from the interpretation of the parse result to the output of the query results.

[49] The DRSXML2Tree class 415 creates a tree from a parsing result. That tree is then processed by the DRSTree2JPQL class 420. The DRSTree2JPQL class 420 is a central place where the parsing result is interpreted. The input is the tree which was created by the DRSXML2Tree class 415. The class instance investigates each tree element and tries to find corresponding objects and relations in the data schema and recognize the semantics of the element in the query context. The functions for the mapping recognition are provided by Mapper class 435. Those functions are grouped in the interfaces corresponding to their functionality. Mapping is described in more detail below.

[50] During the processing of tree elements, the interpreted information is saved in the QueryBuilder Instance 431. After all tree elements are interpreted, QueryBuilder creates a JPQL query on the basis of this information.

[51] Because there is no expression in JPQL that has the same meaning as "contains" in SQL, a work-around for the text search is necessary. If a text search is required, then it will be defined using the "like" function with a particular pattern in the value part. The JPQL query is then translated to SQL, and "like" with the pattern for "contains" is replaced through SQL "contains." The function "like" is also used for the queries with substring search. This simple "like" does not require additional actions and will be ignored during post-processing.

[52] The JPQLToSQLTranslator class 425 transforms a JPQL query to a SQL query using standard Hibernate functionality.

[53] The QueryContainer class 430 saves an incrementally interpreted user request and at the end generates a JPQL query. The QueryContainer class 430 implements QueryBuilder interface 431, which summarizes all for the communication with the class responsible methods.

[54] The QueryContainer class 430 creates the JPQL query, which addresses a particular data structure comprising of JAVA classes. In the QueryContainer class, each requested class is represented through an EntityRequest instance 432 and QueryContainer manages those instances. An EntityRequest object is generated on request if it does not yet exist for the particular class name; otherwise, an existing object is provided. Each EntityRequest instance 432 saves all in the query about the particular class defined information except filter conditions for class attributes. Filter conditions are handled separately in the tree structure, but each filter condition has a reference to the corresponding EntityRequest instance.

[55] Filter conditions are handled in a tree and Tree nodes implements the WherePart interface. Two classes, FilterCondition and FilterContainer, implement the WherePart interface. FilterCondition saves information about the particular filter, and FilterContainer contains information about children nodes. QueryContainer class 430 has reference only to the root of the tree.

[56] Because there were some bugs in the Hibernate Library, and some functionality was missing, some work-arounds were created. Those are implemented by Mapper class 435 and used by QueryContainer 430. The methods are summarized in WorkAroundsForJpqlQuery interface 455, which is described below.

[57] The Mapper class 435 provides different functionality to several components. Mapper implements three interfaces: the RelationMapper interface 440, the TextAndSubstringSearch 445, and the RangeConditionRecognizer 450. Each of them describes a particular functionality. The RelationMapper interface 440 declares two methods. One method maps prepositions to filter expressions; e.g., "from" to "=>". The other method tries to map phrases, separate words or parts of them to elements and relations between them in data structure. Data for the mapping is either defined manually or is loaded from the database.

[58] The TextAndSubstringSearch interface 445 summarizes methods and variables that enable text searching with the SQL function "contains." The RangeConditionRecognizer interface 450 describes two approaches for recognition of affected elements for range filter conditions. Both approaches are used consecutively.

The first approach attempts to identify affected elements in the data structure on the basis of manually-defined mapping with keywords. If the mapping cannot be found in this way, the second approach attempts to find affected elements based on type.

[59] The inventors have overcome two problems contained in the Hibernate Library. Specifically, the WorkAroundsForJpqlQuery interface 455 declares methods that provide work-arounds for those problems.

[60] The first problem is that, if the variable that represents the whole class is used in group by part, then Hibernate automatically uses all to the current class mapped table columns. Apart from performance problems, this causes problems if one of the columns cannot be used in group by (e.g. column has type Text). Moreover, if the whole object is requested, we want to deliver only a subset of all mapped columns. Therefore, the Mapper object 435 provides the list of the default columns for each class name. These default columns are used in the group by and select part of the query if the whole object is requested.

[61] The second problem is that Hibernate is not able to recognize that the same class is joined more than one time. If class a is joined with b and c with a, then Hibernate creates a cross product: $ab*ca$ and it is not possible to say that there is only one a. If such a situation arises, a join is created between two a's again in order to get the right answer. The interface describes a method which delivers a primary key for each class, and therefore enables the creation of those joins.

[62] The QueryContainer class 430 uses the EntityRequestOrganizer component 460 to link EntityRequest objects 432 with each other. Each EntityRequest object corresponds to a requested class and EntityRequestOrganizer create joins between them. If a direct join between classes is not possible, EntityRequestOranizer tries to create joins indirectly through child classes or alternatively through common direct or indirect parents (classes that reference particular classes). The EntityRequestOrganizer component 460 uses ClassRelationsTree 465 to get information about how the classes are related to each other.

[63] The ClassRelationTree class 465 creates a tree from the class data structure and represents how the classes reference each other. Each node saves information about all its children and all its parents. In order to be able to create a tree and not a graph, ClassRelationTree provides the possibility to exclude some references. If two classes can be referenced in alternative ways, the shortest reference will be chosen if the length of the references is different.

[64] Method

[65] An exemplary method for managing a service analytics database containing equipment service information is illustrated by the flow chart 500 shown in FIG. 5. Domain entities from the database are initially mapped, in block 510, to an entity relation model having elements including machines representing single pieces of equipment; at least one of time-stamped events relating to the machines and sensor values relating to the events; and service notifications relating to the machines and including descriptions of services performed. The domain entities may also include spare parts records relating to the machines and relating to the service notifications. Those particular entities, as well as the relations between them as described above with reference to FIG.

2, have been found by the inventors to adequately represent service analytics models describing systems as diverse as medical scanners, power plants, gas turbines and trains.

[66] A query is then received in block 520 containing natural language elements. The query may use one or more alternative approaches, including keywords and phrases, controlled English and full English. The query is parsed in block 530 to extract concepts. Parsing is performed using a parser appropriate for the query approach: semantic parsers for full English, a specialized parser such as an ACE parser for controlled English, and evidence-based analysis for keywords and phrases. Alternatively, an ensemble approach may be used, wherein multiple parsers are used, resulting parses are scored, and one or more parsers are chosen based on the scores.

[67] The concepts are interpreted in block 540 in light of the entity relation model. The query is then translated in block 550 to a standard query language such as SQL using the interpretation of the concepts.

[68] System

[69] The elements of the methodology as described above may be implemented in a computer system comprising a single unit or a plurality of units linked by a network or a bus. An exemplary system 600 is shown in FIG. 6.

[70] A computing apparatus 610 may be a mainframe computer, a desktop or laptop computer or any other device or group of devices capable of processing data. The computing apparatus 610 receives data from any number of data sources that may be connected to the apparatus. For example, the computing apparatus 610 may receive input from a user via an input/output device 648, such as a computer or a computing terminal.

The input/output device includes an input that may be a mouse, network interface, touch screen, etc., and an output that may be a visual display screen, a printer, etc. Input/output data may be passed between the computing apparatus 610 and the input/output device 648 via a wide area network such as the Internet, via a local area network or via a direct bus connection. The computing apparatus 610 may be configured to operate and display information by using, e.g., the input/output device 648 to execute certain tasks. In one embodiment, queries are initiated via the input/output device 648, and query results are displayed to the user via the same device.

[71] The computing apparatus 610 includes one or more processors 620 such as a central processing unit (CPU) and further includes a memory 630. The processor 620, when configured using software according to the present disclosure, includes modules that are configured for performing one or more methods for managing a service analytics database, as discussed herein. Those modules include a database mapping module 624 that performs the function of organizing a service analytics database 650 according to an entity relation model 660. That task is performed at system development, before any queries are received, to create a database representation 670. The database representation 670 may include, for example, a domain vocabulary, a database schema and additional resources. The database mapping module 624 may also perform updating tasks for updating the database representation 670 as the service analytics database 650 changes.

[72] The modules included in the processor 620 also include a query processing module 622 that receives queries from the input/output device 648, performs parsing, interpreting and translating operations as described above, runs the translated query on the database representation 670, and transmits query results to the input/output device 648.

[73] The memory 630 may include a random access memory (RAM) and a read-only memory (ROM). The memory may also include removable media such as a disk drive, tape drive, memory card, etc., or a combination thereof. The RAM functions as a data memory that stores data used during execution of programs in the processor 620; the RAM is also used as a program work area. The ROM functions as a program memory for storing a program executed in the processor 620. The program may reside on the ROM or on any other tangible or non-volatile computer-readable media 640 as computer readable instructions stored thereon for execution by the processor to perform the methods of the invention. The ROM may also contain data for use by the program or by other programs.

[74] Generally, program modules 622, 624 described above include routines, objects, components, data structures and the like that perform particular tasks or implement particular abstract data types. The term "program" as used herein may connote a single program module or multiple program modules acting in concert. The disclosure may be implemented on a variety of types of computers, including personal computers (PCs), hand-held devices, multi-processor systems, microprocessor-based programmable consumer electronics, network PCs, mini-computers, mainframe computers and the like. The disclosed technique may also be employed in distributed computing environments, where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, modules may be located in both local and remote memory storage devices.

[75] An exemplary processing module for implementing the methodology above may be hardwired or stored in a separate memory that is read into a main memory of a processor or a plurality of processors from a computer readable medium such as a ROM or other type of hard magnetic drive, optical storage, tape or flash memory. In the case of a program stored in a memory media, execution of sequences of instructions in the module causes the processor to perform the process steps described herein. The embodiments of the present disclosure are not limited to any specific combination of hardware and software and the computer program code required to implement the foregoing can be developed by a person of ordinary skill in the art.

[76] The term "computer-readable medium" as employed herein refers to any tangible machine-encoded medium that provides or participates in providing instructions to one or more processors. For example, a computer-readable medium may be one or more optical or magnetic memory disks, flash drives and cards, a read-only memory or a random access memory such as a DRAM, which typically constitutes the main memory. Such media excludes propagated signals, which are not tangible. Cached information is considered to be stored on a computer-readable medium. Common expedients of computer-readable media are well-known in the art and need not be described in detail here.

[77] Conclusion

[78] The above describes a general architecture for developing a natural language query capability for service analytics domains. As discussed, a number of specific methods that can be used to implement such architecture. The approaches rely

on a generic high-level ontology common to all service analytics domains; that ontology is also presented above. The high-level ontology must be supplemented with domain-specific details for target applications. However, the remainder of the approach, including question understanding and translation aspects, is general and independent of the domain information. Such architecture and approach have significant benefits. First, the development costs for new domains are lower than the development costs of a complete custom application, due to reuse of most of the code, including the user interface. Second, the approach enables users to ask variety of questions directly, decreasing the need for maintenance and "tweaking" of the application. The architecture is particularly useful for organizations involved in service analytics or condition monitoring on fleets of products.

[79] The foregoing detailed description is to be understood as being in every respect illustrative and exemplary, but not restrictive, and the scope of the disclosure herein is not to be determined from the description, but rather from the claims as interpreted according to the full breadth permitted by the patent laws. It is to be understood that various modifications will be implemented by those skilled in the art, without departing from the scope and spirit of the disclosure.

Claims

What is claimed is:

1. A method for querying a service analytics database, comprising:

mapping domain entities from the database to an entity relation model having elements including:

machines representing single pieces of equipment;

at least one of time stamped events relating to particular machines and sensor values relating to particular times; and

service notifications relating to particular machines and including descriptions of services performed;

receiving a query containing natural language elements;

by a processor, parsing the query to extract concepts;

by a processor, mapping the concepts to the elements of the entity relation model to create an interpretation of the concepts; and

by a processor, translating the query to a standard query language using the interpretation of the concepts.

2. A method as in claim 1, wherein the elements of the entity relation model further include:

spare parts records relating to particular machines and relating to the service notifications.

3. A method as in claim 1, wherein the query is formulated as an unstructured set of keywords and short phrases.

4. A method as in claim 3, wherein the parsing is a semantic parsing applied to extract additional keywords from the short phrases, and to extract pieces of evidence indicating type of question and involved entities in light of the entity relation model.

5. A method as in claim 1, wherein the query is formulated as a predefined subset of actual language constructed according to a set of rules.

6. A method as in claim 5, wherein the parsing uses a parser limited to parsing the subset of actual language.

7. A method as in claim 1, wherein the query is formulated as an unconstrained query in an actual language.

8. A method as in claim 7, wherein the parsing further comprises constructing syntactic parse trees of the input, and the method further comprises:

using a semantic role labeler to identify key items and relationships.

9. A method as in claim 1, wherein the standard query language is a resource descriptive framework language.

10. A method as in claim 1, wherein the standard query language is structured query language.

11. A method as in claim 1, wherein parsing the query to extract concepts further comprises:

using multiple question understanding modules in parallel to produce multiple parsing results;

assigning confidence scores to the parsing results; and

selecting the parsing result having the highest confidence score.

12. A non-transitory computer-readable medium having stored thereon computer readable instructions for querying a service analytics database, wherein execution of the computer readable instructions by a processor causes the processor to perform operations comprising:

mapping domain entities from the database to an entity relation model having elements including:

machines representing single pieces of equipment;

at least one of time-stamped events relating to particular machines and sensor values relating to particular times; and

service notifications relating to particular machines and including descriptions of services performed;

receiving a query containing natural language elements;

parsing the query to extract concepts;

mapping the concepts to the elements of the entity relation model to create an interpretation of the concepts; and translating the query to a standard query language using the interpretation of the concepts.

13. A non-transitory computer-readable medium as in claim 12, wherein the elements of the entity relation model further include:

spare parts records relating to particular machines and relating to the service notifications.

14. A non-transitory computer-readable medium as in claim 12, wherein the query is formulated as an unstructured set of keywords and short phrases.

15. A non-transitory computer-readable medium as in claim 14, wherein the parsing is a semantic parsing applied to extract additional keywords from the short phrases, and to extract pieces of evidence indicating type of question and involved entities in light of the entity relation model.

16. A non-transitory computer-readable medium as in claim 12, wherein the query is formulated as a predefined subset of actual language constructed according to a set of rules.

17. A non-transitory computer-readable medium as in claim 16, wherein the parsing uses a parser limited to parsing the subset of actual language.

18. A non-transitory computer-readable medium as in claim 12, wherein the query is formulated as an unconstrained query in an actual language.

19. A non-transitory computer-readable medium as in claim 18, wherein the parsing further comprises constructing syntactic parse trees of the input, and the method further comprises:

using a semantic role labeler to identify key items and relationships.

20. A non-transitory computer-readable medium as in claim 12, wherein the standard query language is a resource descriptive framework language.

2 1. A non-transitory computer-readable medium as in claim 12, wherein the standard query language is structured query language.

22. A non-transitory computer-readable medium as in claim 12, wherein parsing the query to extract concepts further comprises:

using multiple question understanding modules in parallel to produce multiple parsing results;

assigning confidence scores to the parsing results; and

selecting the parsing result having the highest confidence score.

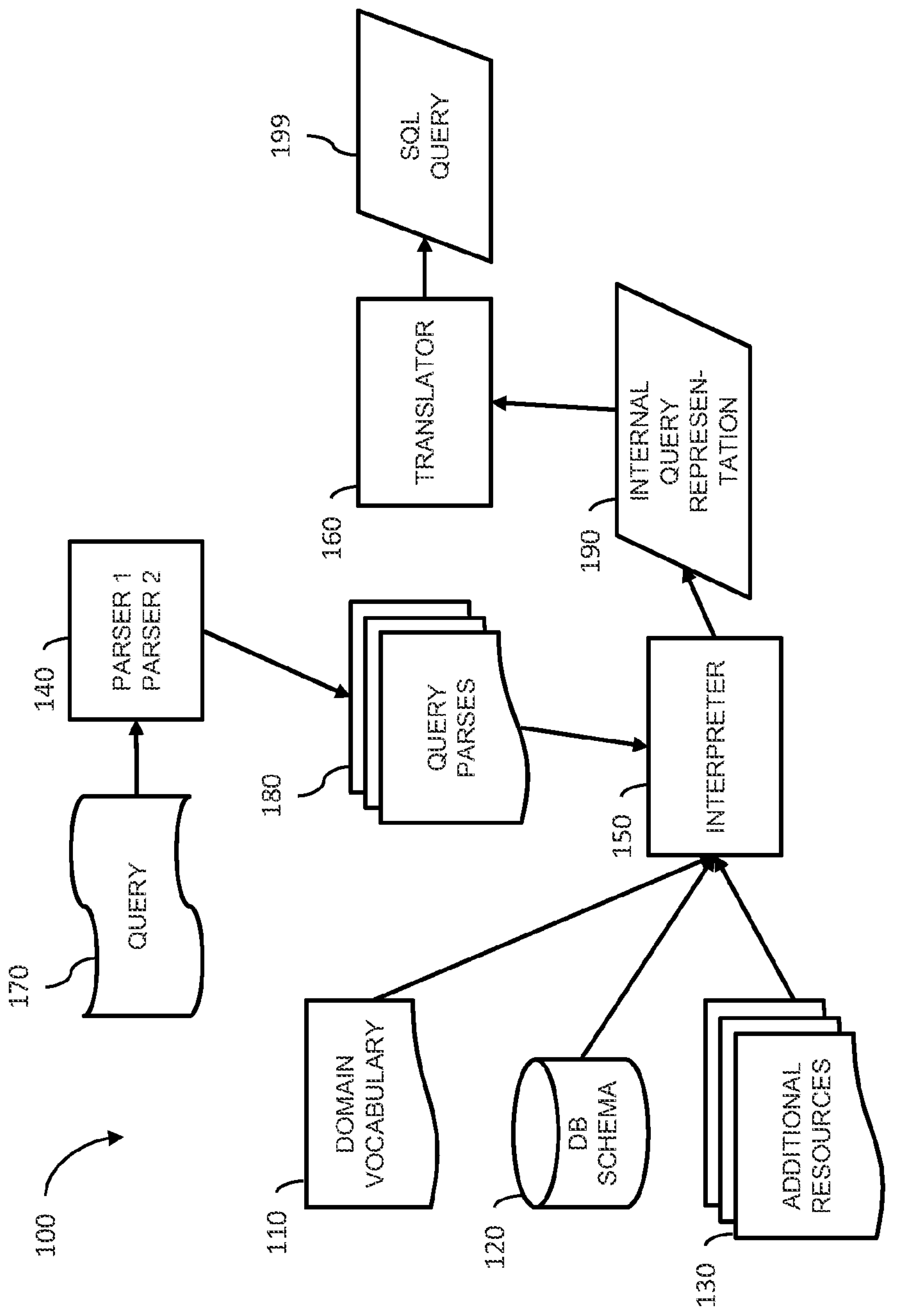
100
170
QUERY
140
PARSER 1
PARSER 2
180
QUERY PARSES
150
INTERPRETER
110
DOMAIN VOCABULARY
120
DB SCHEMA
130
ADDITIONAL RESOURCES
190
INTERNAL QUERY REPRESEN-TATION
160
TRANSLATOR
199
SQL QUERY

Fig. 1

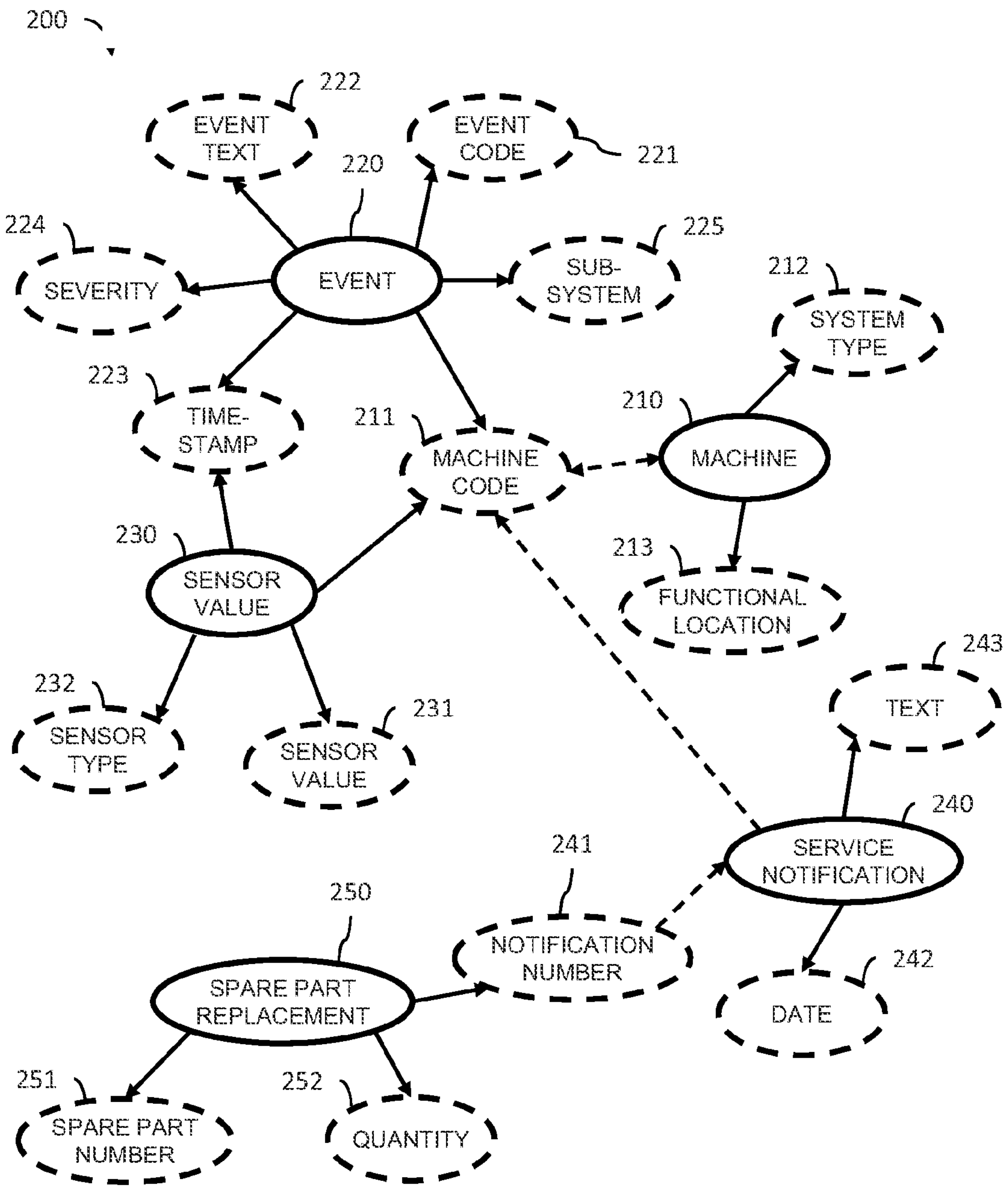
200
EVENT TEXT
222
EVENT CODE
221
220
EVENT
224
SEVERITY
SUB-SYSTEM
225
212
SYSTEM TYPE
223
TIME-STAMP
211
MACHINE CODE
210
MACHINE
230
SENSOR VALUE
213
FUNCTIONAL LOCATION
232
SENSOR TYPE
231
SENSOR VALUE
243
TEXT
240
SERVICE NOTIFICATION
241
NOTIFICATION NUMBER
250
SPARE PART REPLACEMENT
242
DATE
251
SPARE PART NUMBER
252
QUANTITY

Fig. 2

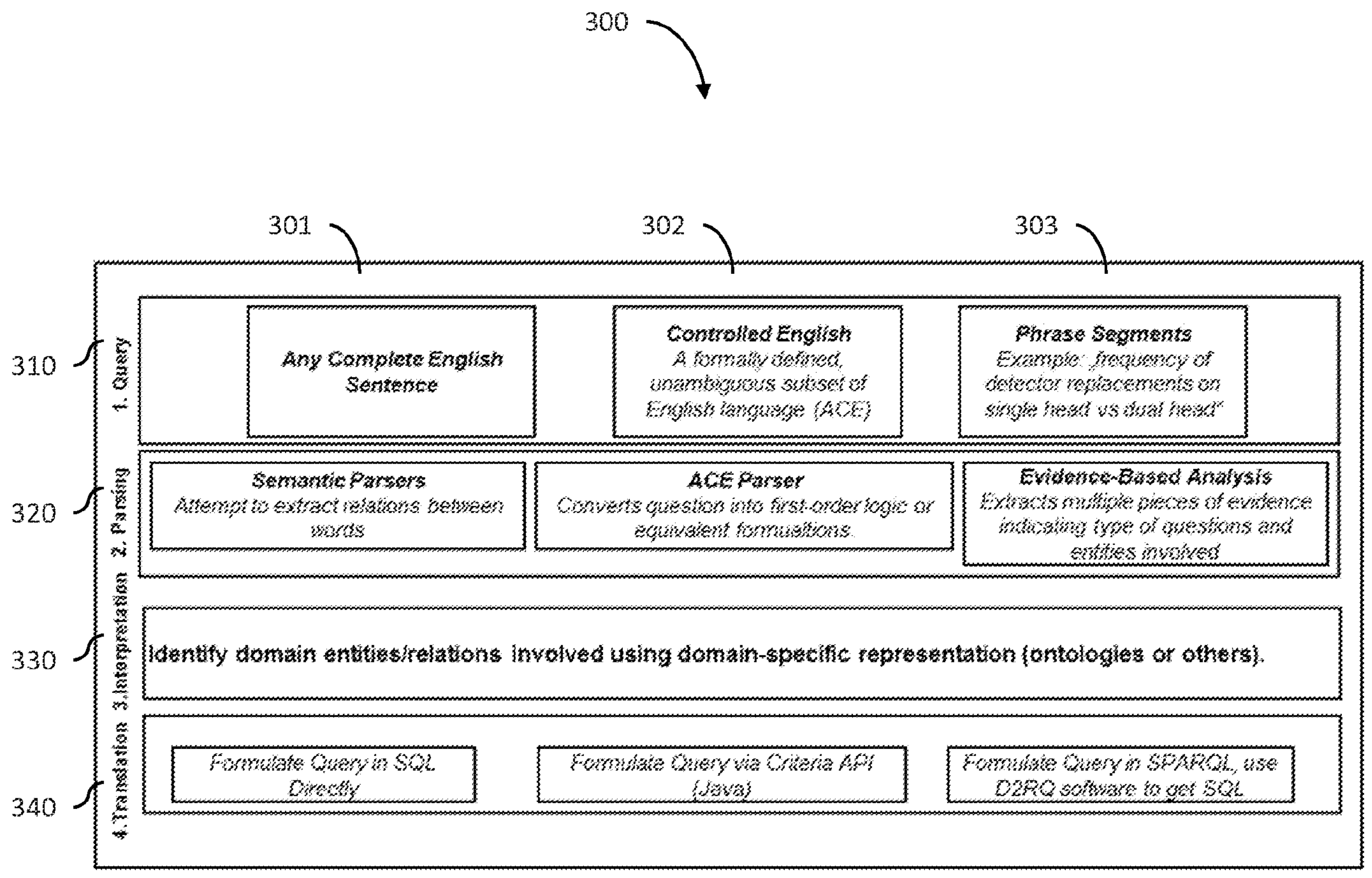
300
301
302
303
310
320
330
340
1. Query
2. Parsing
3.Interpretation
4.Translation
Any Complete English Sentence
Controlled English
A formally defined, unambiguous subset of English language (ACE)
Phrase Segments
Example: „frequency of detector replacements on single head vs dual head"
Semantic Parsers
Attempt to extract relations between words
ACE Parser
Converts question into first-order logic or equivalent formulations.
Evidence-Based Analysis
Extracts multiple pieces of evidence indicating type of questions and entities involved
Identify domain entities/relations involved using domain-specific representation (ontologies or others).
Formulate Query in SQL Directly
Formulate Query via Criteria API (Java)
Formulate Query in SPARQL, use D2RQ software to get SQL

Fig. 3

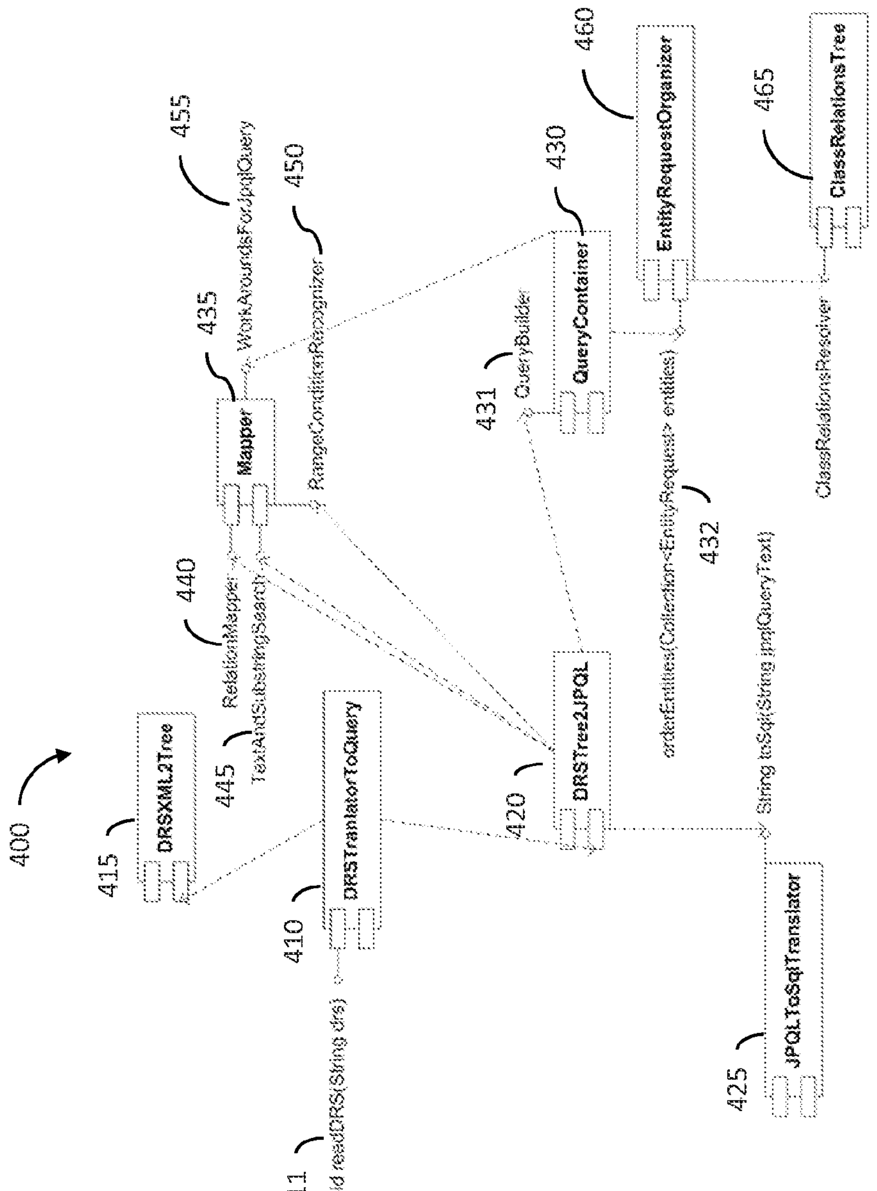
400
415
DRSXML2Tree
410
DRSTranlatorToQuery
411
void readDRS(String drs)
445
TextAndSubstringSearch
440
RelationMapper
420
DRSTree2JPQL
425
JPQLToSqlTranslator
String toSql(String jpqlQueryText)
435
Mapper
RangeConditionRecognizer
450
WorkAroundsForJpqlQuery
455
431
QueryBuilder
QueryContainer
430
432
orderEntities(Collection<EntityRequest> entities)
EntityRequestOrganizer
460
ClassRelationsResolver
ClassRelationsTree
465

Fig. 4

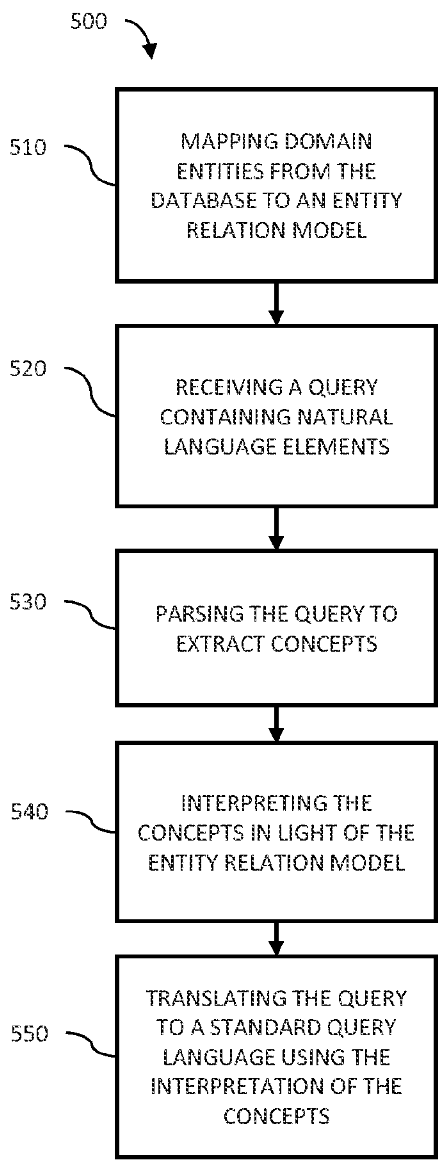
500
510
MAPPING DOMAIN ENTITIES FROM THE DATABASE TO AN ENTITY RELATION MODEL
520
RECEIVING A QUERY CONTAINING NATURAL LANGUAGE ELEMENTS
530
PARSING THE QUERY TO EXTRACT CONCEPTS
540
INTERPRETING THE CONCEPTS IN LIGHT OF THE ENTITY RELATION MODEL
550
TRANSLATING THE QUERY TO A STANDARD QUERY LANGUAGE USING THE INTERPRETATION OF THE CONCEPTS

Fig. 5

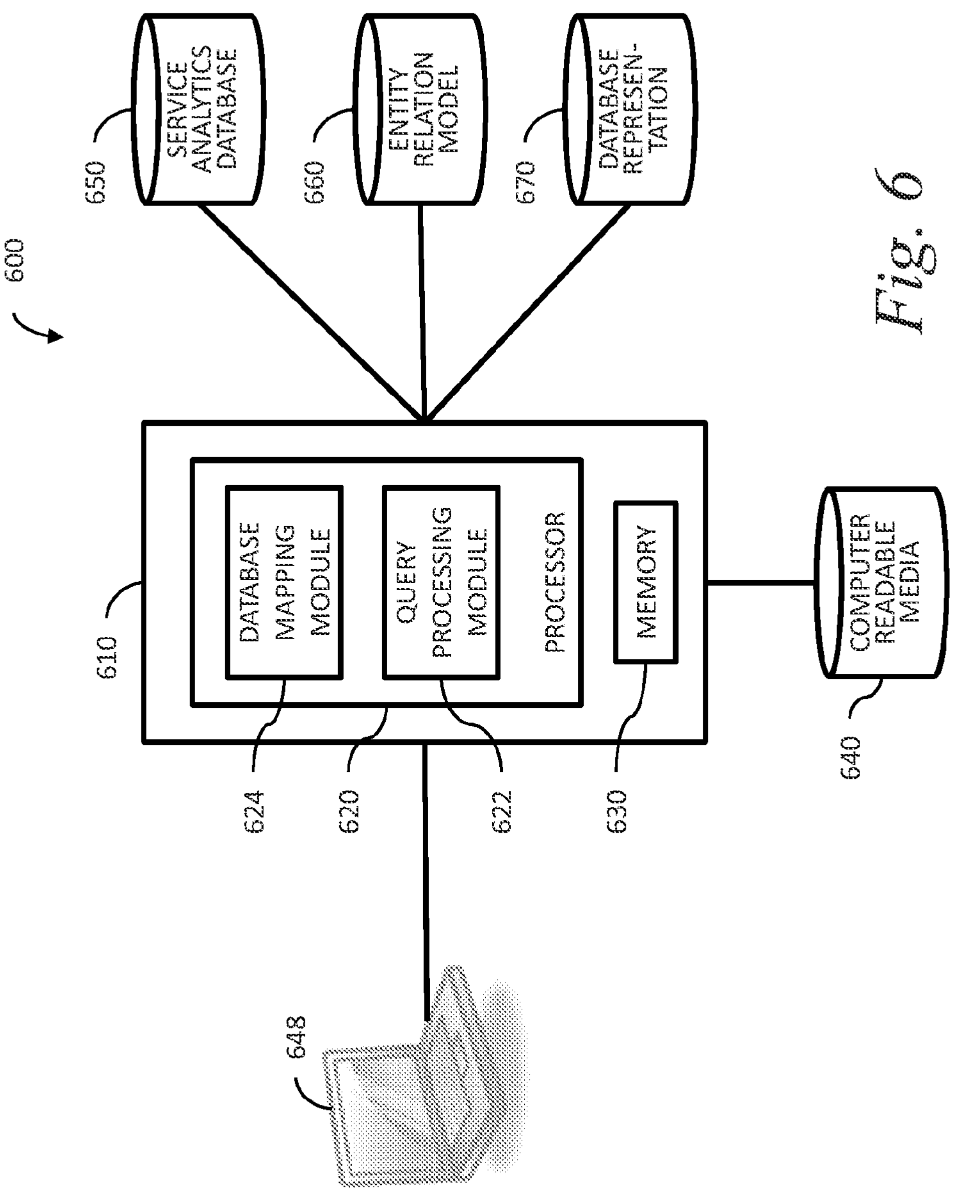
600
650
SERVICE ANALYTICS DATABASE
660
ENTITY RELATION MODEL
670
DATABASE REPRESEN-TATION
610
DATABASE MAPPING MODULE
QUERY PROCESSING MODULE
PROCESSOR
MEMORY
624
620
622
630
COMPUTER READABLE MEDIA
640
648

Fig. 6