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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
3 April 2014 (03.04.2014)

WIPO | PCT

(10) International Publication Number
WO 2014/052873 A1

(51) International Patent Classification:
G06F 9/44 (2006.01)

(21) International Application Number:
PCT/US2013/062369

(22) International Filing Date:
27 September 2013 (27.09.2013)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
61/707,343 28 September 2012 (28.09.2012) US

(71) Applicant: **AB INITIO TECHNOLOGY LLC** [US/US]; 201 Spring Street, Lexington, Massachusetts 02421 (US).

(72) Inventors: **IKAI, Taro**; Ogikubo 3-40-6, Suginami-ku, Tokyo 167-0051 (JP). **ANDERSON, Arlen**; 3 Lower Street, Islip, Kidlington OX5 2SG (GB).

(74) Agent: **GERRATANA, Frank L.**; Fish & Richardson P.C., P.O. Box 1022, MInneapolis, Minnesota 55440-1022 (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, 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, SA, 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, KM, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))*

(54) **Title**: GRAPHICALLY REPRESENTING PROGRAMMING ATTRIBUTES

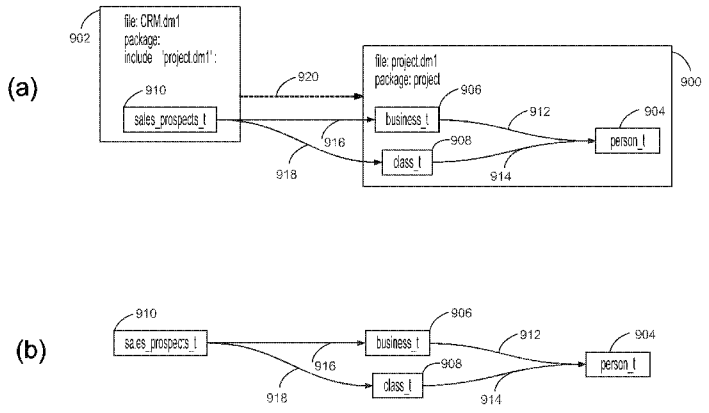

FIG. 9

(57) **Abstract**: A computing system for representing information, the computing system including at least one processor configured to process information. The processing includes defining a data structure representing a hierarchy of at least one programming attribute for developing an application. The data structure is stored in a file to allow the data structure to be used by other data structures stored in other files. The processing also includes producing a visual diagram including a graphical representation of the data structure and a graphical representation of the file storing the data structure. The visual diagram also includes a graphical representation of a relationship between the data structure and another data structure and a graphical representation of a relationship between the file storing the data structure and another file storing the other data structure. The computing system also including an output device for presenting the visual diagram that includes the graphical representations of the data structure and file and the graphical representations of the relationship of the data structures and the relationship of the files.

WO 2014/052873 A1

GRAPHICALLY REPRESENTING PROGRAMMING ATTRIBUTES

CLAIM OF PRIORITY

This application claims priority under 35 USC §119(e) to U.S. Patent Application Serial No. 61/707,343, filed on September 28, 2012, the entire contents of which are hereby incorporated by reference.

BACKGROUND

This description relates to a graph based approach to representing programming attributes.

Complex computations can often be expressed as a data flow through a directed graph (called a "dataflow graph"), with components of the computation being associated with the vertices of the graph and data flows between the components corresponding to links (arcs, edges) of the graph. The components can include data processing components that receive data at one or more input ports, process the data, and provide data from one or more output ports, and dataset components that act as a source or sink of the data flows. A system that implements such graph-based computations is described in U.S. Patent 5,966,072, EXECUTING COMPUTATIONS EXPRESSED AS GRAPHS. Various types of data may be received, processed and output by the components of the graph. Due to similar processing functionality, equivalent types of data may be used and re-used for different applications.

SUMMARY

In one aspect, a method for representing information includes defining a data structure representing a hierarchy of at least one programming attribute for developing an application. The data structure is stored in a file to allow the data structure to be used by other data structures stored in other files. The method also includes producing a visual diagram including a graphical representation of the data structure and a graphical representation of the file storing the data structure. The visual diagram also includes a graphical representation of a relationship between the data structure and another data structure and a graphical representation of a relationship between the file storing the data structure and another file storing the other data structure.

In another aspect, a computer-readable storage medium stores a computer program for representing information. The computer program includes instructions for causing a computing system to define a data structure representing a hierarchy of at least one programming attribute for developing an application. The data structure is stored in a file to allow the data structure to be used by other data structures stored in other files. The instructions also cause the computing system to produce a visual diagram including a graphical representation of the data structure and a graphical representation of the file storing the data structure. The visual diagram also includes a graphical representation of a relationship between the data structure and another data structure and a graphical representation of a relationship between the file storing the data structure and another file storing the other data structure.

In another aspect, a computing system for representing information includes at least one processor configured to process information. The processing including defining a data structure representing a hierarchy of at least one programming attribute for developing an application. The data structure is stored in a file to allow the data structure to be used by other data structures stored in other files. The processing also including producing a visual diagram including a graphical representation of the data structure and a graphical representation of the file storing the data structure. The visual diagram also includes a graphical representation of a relationship between the data structure and another data structure and a graphical representation of a relationship between the file storing the data structure and another file storing the other data structure. The computer system also includes an output device for presenting the visual diagram that includes the graphical representations of the data structure and the file and the graphical representations of the relationship of the data structures and the relationship of the files.

In another aspect, a computing system for representing information includes means for processing that include defining a data structure representing a hierarchy of at least one programming attribute for developing an application. The data structure is stored in a file to allow the data structure to be used by other data structures stored in other files. The processing also includes producing a visual diagram including a graphical representation of the data structure and a graphical representation of the file storing the data structure. The visual diagram also includes a graphical representation of a relationship between the data structure and another data structure and a graphical

representation of a relationship between the file storing the data structure and another file storing the other data structure. The computer system also includes means for presenting the visual diagram that includes the graphical representations of the data structure and the file and the graphical representations of the relationship of the data structures and the relationship of the files.

Implementations may include any or all of the following features. The graphical representation of the files may be removable to allow for manipulating the graphical representations of the data structure to define one or more data structure groups and to create one or more new files for storing the one or more data structure groups. The defined data structure may be manipulated to adjust the at least one programming attribute. The defined data structure may be manipulated for inserting the at least one programming attribute into a file. A statement may be inserted with the at least one programming attribute into the file. Manipulating the data structure may include a drag and drop operation. Manipulating the defined data structure may include at least one of adding, deleting and editing content of the hierarchy of the programming attribute. The programming attribute may be a named data type, a function, etc. The relationship between the data structure and the other data structure may represent lineage of the programming attribute.

Aspects can include one or more of the following advantages.

Graphically representing programming attributes (e.g., fields, named data type structures, functions, etc.) allows a developer to relatively quickly ascertain attribute details (e.g., data types being used) and relationships among attributes (e.g., a named data type structure using previously defined fields) and files used to store the attributes. Presented in graphical form, attributes, files storing the attributes, etc. may be efficiently manipulated, for example, to edit named data type structures and thereby allow the changes to propagate as needed.

Other features and advantages of the invention will become apparent from the following description, and from the claims.

DESCRIPTION OF DRAWINGS

FIG. 1 is a block diagram of a system for executing graph-based computations.

FIGs. 2-6 are user interfaces presenting data type information.

FIGs. 7-12 are user interfaces presenting graphical representations of data type information.

FIG. 13 is a flowchart of an exemplary programming attribute presentation procedure.

DESCRIPTION

FIG. 1 shows an exemplary data processing system 100 in which programming attributes such as data types, executable functions, etc. may be graphically presented, for example, to allow a casual viewer to efficiently determine the content, hierarchy and lineage of the attributes. In general, to provide such functionality, the system 100 includes a data source 102 that may include one or more sources of data such as storage devices or connections to online data streams, each of which may store data in any of a variety of storage formats (e.g., database tables, spreadsheet files, flat text files, or a native format used by a mainframe). In this example, an execution environment 104 includes a pre-processing module 106 and an execution module 112. The execution environment 104 may be hosted on one or more general-purpose computers under the control of a suitable operating system, such as the UNIX operating system. For example, the execution environment 104 can include a multiple-node parallel computing environment including a configuration of computer systems using multiple central processing units (CPUs), either local (e.g., multiprocessor systems such as SMP computers), or locally distributed (e.g., multiple processors coupled as clusters or MPPs), or remote, or remotely distributed (e.g., multiple processors coupled via a local area network (LAN) and/or wide-area network (WAN)), or any combination thereof.

The pre-processing module 106 reads data from the data source 102 and performs corresponding processing operations, e.g., in anticipation of further proceeding by other modules. Storage devices providing the data source 102 may be local to the execution environment 104, for example, being stored on a storage medium connected to a computer running the execution environment 104 (e.g., hard drive 108), or may be remote to the execution environment 104, for example, being hosted on a remote system (e.g., mainframe 110) in communication with a computer running the execution environment 104, over a remote connection.

The execution module 112 uses the processed data generated by the pre-processing module 106, for example, to process the data 114 (e.g., enterprise data,

company records, etc.) stored in a data storage system 116 accessible to the execution environment 104. The data storage system 116 is also accessible to a development environment 118 in which a developer 120 is able to review, edit, etc. the information stored in the data storage system 116. In some arrangements, the development environment 118 may be utilized in preparing and adjusting the execution environment 104 for performing the desired operations. For example, the development environment 118 may be a system for developing applications as dataflow graphs that include vertices (representing components or datasets) connected by directed links (representing flows of work elements) between the vertices. For example, such an environment is described in more detail in U.S. Publication No. 2007/0011668, entitled "Managing Parameters for Graph-Based Applications," incorporated herein by reference. A system for executing such graph-based computations is described in U.S. Patent 5,566,072, EXECUTING COMPUTATIONS EXPRESSED AS GRAPHS, incorporated herein by reference. Dataflow graphs made in accordance with this system provide methods for getting information into and out of individual processes represented by graph components, for moving information between the processes, and for defining a running order for the processes. This system includes algorithms that choose interprocess communication methods (for example, communication paths according to the links of the graph can use TCP/IP or UNIX domain sockets, or use shared memory to pass data between the processes).

The pre-processing module 106 can receive data from a variety of types of systems including different forms of database systems. The data may be organized as records having values for respective fields (also called "attributes" or "columns"), including possibly null values. When first reading data from a data source, the pre-processing module 106 typically starts with some initial format information about records in that data source. In some circumstances, the record structure of the data source may not be known initially and may instead be determined after analysis of the data source. The initial information about records can include the number of bits that represent a distinct value, the order of fields within a record, and the type of data (e.g., string, signed/unsigned integer) represented by the bits.

Similar to being used in a record, different data types (e.g., strings, integers, flowing point values) can be used for processing data (e.g., records) performed by the execution environment 104, the development environment 118, etc. For example,

various types of data types may be defined and used for processing records (e.g., employee records, student records, etc.). To process such records, individual primitive data types (e.g., strings, date, datetime, integer, decimal, etc.) may be used in concert to define an organization of data types, referred to as a data type structure. For example, the following record groups four primitive types to represent attributes of a person:

```
record
  string(",") surname;
  string(",") given_name;
  date("YYYY-MM-DD") date_of_birth;
  integer(1) gender; // 0 for male, 1 for female
  decimal(9) SSN;
end
```

The record includes a number of fields (e.g., the first and last name of the individual is represented along with their date of birth, gender and social security number), and each field consists of a name and a data type. Referring to FIG. 2, such a record may be defined by an editor. In this example, a user interface 200 presents an editor for defining the format of data being read from an input file. Along with defining the data being read (e.g., surname, date of birth, gender), the individual data types (e.g., string, date, integer) and limitations for each field are defined. While this example presents the record type as including five distinct entries of information, the record type may be expanded or contracted. For example, more data may be appended, deleted, combined, etc. with current data. Continuing from the instructions above, a record type may be nested, for example, to express a business entity and a list of corresponding employees:

```
/* Business entity */
record
    string(",") business_name;
    decimal(9) tax_payer_id;
    decimal(11) main_TEL_no;
    record
        record
```

```
            string(",") last; // last name
            string(",") first; // first name
        end name; // subrecord
        record
            date("YYYY-MM-DD") date_of_birth;
            integer(1) gender; // 0 for male, 1 for female
        end bio; // subrecord
        decimal(9) SSN;
    end[integer(4)] employees; // vector of employees
end
```

Through creating and naming data type structures, equivalent fields, etc. may be used for more than one application (e.g., to represent student information, ordering information for employees of a number of companies, etc.). Referring to FIG. 3, a user interface 300 may be implemented for presenting a data type structure that includes nested data type information. In this arrangement, fields associated with a company are defined in an upper portion (as represented by a bracket 302) and fields for the employee information are defined in a lower portion (as represented by bracket 304). As can be imagined, data type structures may be defined for a large variety of applications and many similar fields may be used and re-used (e.g., to define a record for student information, a record for employee information, etc.).

Referring to FIG. 4, one or more techniques may be implemented to assist a developer with appropriately defining fields and grouping the fields to form data type structures for use, e.g., in application development. For example, by interacting with a user interface 400 (e.g., selecting a field with a pointing device), a drop-down menu 402 may appear and present a listing of named data type structures that may be potentially selected (by the developer) for use in an application. As represented in the drop-down menu 402, as more and more named data type structures are defined, the presented listing may become onerously large and difficult for a user to navigate to identify a named data type structure of interest. For example, along with frequently selected named data type structures, the listing may grow over time to include a large number of less-frequently-used named data type structures that are unique to specific applications. As illustrated in the figure, the sheer number of entries in the menu 402

could drastically affect a developer's efficiency and may even discourage use of the user interface 400. Further, only being assigned names and lacking any other information (as shown in the entries of the menu 402) many named data type structures may be redundant and provide no information regarding relationships (if any) among other named data type structures.

One or more techniques and methodologies may be implemented to provide more manageable representations of data type structures. In one example, data type structures used in multiple, different applications may be commonly defined. From the example above, the data type structure for each student is equivalent to the data type structure of each business employee and can be commonly defined as:

```
record
  record
    string(",") last; // last name
    string(",") first; // first name
  end name; // subrecord
  record
    date("YYYY-MM-DD") date_of_birth;
    integer(1) gender; // 0 for male, 1 for female
  end bio; // subrecord
  decimal(9) SSN;
end
```

By naming this common data type structure (e.g., "person_t"), the named data type structure can be called out and used in applications for similar purposes such as collecting information for student and employee applications. For example, the named data type structure ("person_t") may be defined as:

```
type person_t =
record
 record
  string(",") last; // last name
  string(",") first; // first name
 end name; // subrecord
 record
```

```
    date("YYYY-MM-DD") date_of_birth;
    integer(1) gender; // 0 for male, 1 for female
  end bio; // subrecord
  decimal(9) SSN;
end;
```

Once defined, the named data type structure ("person_t") can be used to respectively define a data type structure for the business employee and classroom student applications:

```
/* Business entity */
record
  string(",") business_name;
  decimal(9)  tax_payer_id;
  decimal(11) main_TEL;
  person_t[integer(4)] employees; // here
end

/* Classroom */
record
  string(",") home_room_instructor;
  decimal(2) grade;
  person_t[integer(4)] students; // and here
end;
```

Similarly, the new data type structures may be named for use in multiple applications. For example, a named data type structure (titled "business_t") that uses the named data type structure "person_t" may be defined for storing business information:

```
type business_t =
record
  string(",") business_name;
  decimal(9) tax_payer_id;
```

```
  decimal(11) main_TEL;
  person_t[integer(4)] employees; // vector of employees
end;
```

Similarly, a named data type structure (named "class_t") that uses the named data type structure "person_t" may be defined for storing class information:

```
type class_t =
record
  string(",") home_room_instructor;
  decimal(2) grade;
  person_t[integer(4)] students; // vector of students
end;
```

Various advantages may be provided from using a common data type structure to define multiple other data type structures. For example, rather than needing to individually adjust the two data type structures (to adjust an included data type structure), the commonly used named data type structure may be redefined once. Correspondingly, by adjusting the named data type structure, any changes would be reflected in both of the data type structures (and any other data type structures that use the named data type structure). Thereby, efficiency may be improved by allowing developers to adjust a single instance of a named data type structure being used in multiple data structures. However, developers should remain alert to such propagating adjustments (e.g., when the developer is interested in only adjusting an included data type structure in less than all of the data type structures that use the included data type structure).

Referring to FIG. 5, once defined, one or more techniques or methodologies may be implemented for storing the named data type structures and retrieving the structures for use. For example, the named data type structures may be expressed in a language such as Data Manipulation Language (DML) and stored in text files (referred to as DML files). Using the named data type structures defined above (e.g., "person_t"), a DML file titled "project.dml" may be stored in a storage device and retrieved for using the defined data type structure. Various techniques may be

implemented for accessing and using named data type structures stored in DML files. For example, a file (e.g., another DML file) may include one or more instructions (e.g., an "include" statement, a "package" statement, etc.) that allows one or more other DML files (and the named data type structures defined within the file) to be accessed and used. For example, the contents of a DML file may include the following language:

```
include "project.dml";

type sales_prospects_t =
record
  class_t[integer(4)] educational;
  business_t[integer(4)] commercial;
end;
```

By accessing the DML file ("project.dml") through the use of the "include" statement, the named data type structures (e.g., "business_t") defined within the DML file can be retrieved and used to define a data type structure (e.g., "sales_prospects_t") defined within the file. Such a capability reduces redundancy of data type definitions along with the probability of conflicting data type structures (e.g., data type structures with the same name but different type definitions).

As illustrated in the figure, once defined and stored, named data type structures may be presented for selection and use by a developer. For example, a user interface 500 may include programming attributes for building applications (e.g., global variables 502, functions 504, user-defined types 506, etc.). As presented in the user interface 500, three named data type structures (e.g., "person_t", "business_t" and "class_t") are included in the user-defined types 506 and may be selected for use by a developer. However, similar to the drop down menu 402 presented in FIG. 3, as the list of data type structures grows for various applications, the entries included in the user-defined types 506 may become unwieldy along with redundant entries becoming rampant. Through the growth of frequently used and less frequently used named data type structures, challenges may arise to identify what particular data type structures are available. Redundancy may also become an issue as identifying previously-defined data type structures may become difficult and overly time-consuming for a developer.

While the presentation of the user interface 500 provides a listing of each named data type structure, little additional information is provided. For example, lineage of dependency among data type structures, and how the related data types can be grouped, etc. is generally absent from the user interface 500. Further, while the name of the data type structure is present, no information regarding the content of the data type structure is provided from the interface beyond what can be gleamed from the title of each data type structure.

Referring to FIG. 6, a graphical representation 600 is illustrated that represents a named data type structure and the collection of fields that define the structure. Additionally the graphical representation 600 identifies the DML file 602 (e.g. "project.dml") within which the data type structure is defined and stored. A portion of the graphical representation 600 provides an optional name 604 (e.g., DML files) for the file, referred to as a "package", which provides a prefix for the names of the attributes defined in the files, when called external to the file. In this particular example, the package name has been left blank.

In this example, the graphical representation 600 presents the named data type structure "person_t" as being defined as:

```
type person_t =
record
      string(",") surname;
      string(",") given_name;
      date("YYYY-MM-DD") date_of_birth;
      integer(1) gender; // 0 for male, 1 for female
      decimal(9) SSN;
end;
```

In this example, the graphical representation 600 is oriented to be read from the viewer's left to right and initially presents the named data type structure name 606 (e.g., "person_t") on the viewer's left. From the structure defined above, a series of rectangles present the individual fields (e.g., surname 608, given_name 610, date of birth 612, gender 614 and social security number 616) nested into the data type structure "person_t". The graphical representation 600 conveys a hierarchical layout of the named data type structure and its content. Located to the far left side of the graphical representation 600, the name 606 identifies the overall structure while the

individual fields 608, 610, 612, 614, 616 are located more to the right and indicate their residing in a lower level in the hierarchy of the data type structure. From this presentation, the viewer is provided an easy-to-read graphical layout of the information associated with the named data type structure. While rectangle shapes are used in this instance to present the information, other shapes and/or collections of difference shapes may be used for presentations. Other graphical features may also be incorporated for assisting a viewer in efficiently ascertaining the named data type structure information. For example, different colors may be implemented, e.g., to quickly alert a viewer to potential issues (e.g., repeated or conflicting fields being defined within a named data type structure). In this instance, static graphics are used for presenting the information; however, graphics that change over time (e.g., animations, video, etc.) may also be used, for example, to quickly grab a viewer's attention. Other graphical representations may also be used, for example, rather than using a left-to-right reading orientation, other layouts and orientations may be implemented for presenting one or more named data structures.

Referring to FIG. 7, a graphical representation 700 illustrates a DML file (e.g., named "project.dml") within which three named data type structures are defined. In particular, along with the graphical representation 600 of the previously defined named data type structure "person_t" (shown in FIG. 6), two additional named data type structures (titled "business_t" and "class_t") are graphically represented in the DML file:

```
type business_t =
record
  string(",") business_name;
  decimal(9)  tax_payer_id;
  decimal(11) main_TEL;
  person_t[integer(4)] employees; // here
end
```

and

```
type class_t =
record
```

```
    string(",") home_room_instructor;
    decimal(2) grade;
    person_t[integer(4)] students; // and here
  end;
```

Both the "business_t" named data type structure and the "class_t" named data type structure, illustrated by corresponding graphical representations 702, 704, include fields (e.g., the "employee" field included in the "business_t" named data type structure, and the "student" field included in the "class_t" data type structure) that are defined by the named data type structure "person_t". To graphically illustrate use of the "person_t" named data type structure by the "business_t" and the "person_t" named data type structures, corresponding arrowed lines 706, 708 show the connections between the pairs of named data type structures. From these graphically represented relationships, a viewer (e.g., a developer) can relatively quickly identify the relationships between the named data type structures such as information shared among data type structures, lineage of the use of the previously defined named data type structure (such as "person_t"), etc. Along with providing a layout of the relationships of the named data type structures, the graphical representation 700 visually alerts the viewer to potential adjustment issues. For example, if changes are made to the "person_t" named data type structure, as illustrated by the two arrowed lines 706, 708, both the "business_t" and "class_t" named data type structures would be affected by the changes (e.g., the "employees" field of the "business_t" named data type structure and the "students" field of the "class_t" named data type structure would experience any changes to the "person_t" named data type structure). Similar to presenting the relationship among data type structures, other types of relationships may be graphically presented such as relationships between files.

Referring to FIG. 8, two DML files are graphically represented along with their relationships. The graphical representation 700 of the "project.dml" file (shown in FIG. 7) is presented along with the three named data structures defined by the DML file (e.g., the "class_t" named data structure 702, the "business_t" named data structure 704 and the "person_t" named data structure 600). Additionally another DML file titled "CRM.dml" is illustrated with a graphical representation 800 within which a named data type structure (titled "sales_prosects_t") is represented. In this example, two

named data type structures are defined by the "CRM.dml" file (e.g., named "educational" and "commercial") and each of the two named date type structures are defined through named data type structures provided by the "project.dml". In particular, the "educational" named data type structure is defined from the "class_t" named data type structure and the "commercial" named data type structure is defined by the "business_t" named data type structure. To represent these two relationships between the "CRM.dml" file and the "project.dml" file, two arrowed lines 802, 804 are illustrated as linking the two graphical representations of the files. Similar to the representations of the named data type structures being linked within a file, similar relationships may be formed through the linking of fields and named type definitions, etc. between two or more files. For example, adjustments made to the definitions of the "class_t" named data type structure 702 or the "business_t" named data type structure 704 (e.g., by changing the "person_t" named data structure 600) in the "project.dml" file can impact the fields (e.g., "educational" and "commercial") and named data type structures (e.g., "sales_prospect_t" named data type structure) in the linked "CRM.dml" file.

Along with presenting graphical representations for the relationships among the fields and named data type structures of the multiple files, the relationship among the files may be graphically represented to the viewer. For example, file level operations may be represented. In this example, for the fields of the CRM.dml file (e.g., "educational" and "commercial") to attain access to the named data type structures in the "project.dml" file, the "project.dml" file needs to be identified by the "CRM.dml" file. For example, an "include" statement may be entered in the "CRM.dml" file to attain access to the "project.dml" file. To graphically represent the identification, along with listing its own file name 806 in the graphical representation 800, the one or more needed files (e.g., the "project.dml") are also represented as needed packages 808. Further, in this example an arrowed, dashed line 810 graphically represents the file-level operation of the "CRM.dml" file of using an "include" statement to attain access to the contents of the "project.dml" file.

Referring to FIG. 9a, various types of graphical representations may be used to illustrate relationships among files, fields and named data type structures. For example, to reduce the visual complexity of the graphical representations, various amounts of detail may be reduced or removed. In the illustrated example, individual field

information is removed from the graphical representations 900, 902 of the two DML files. By removing the information, graphical representations 904, 906, 908, 910 of the data type structures are compacted by simply representing the names of each named data type structure (e.g., "person_t", "business_t", "class_t" and "sales_prospects_t"). Along with reducing the amount of visual business of the graphical representations, real estate is conserved for other information such as the arrowed lines 912, 914 that represent relationships between named data type structures within a file, and arrowed lines 916, 918 that represent relations between named data type structures that reside in different files. Similarly, the conserved real estate may assist the viewer in quickly recognizing other relationships, e.g., files identifying other files with an arrowed dashed line 920 to indicate the use of an "include" statement to provide file access. Along with providing the viewer (e.g., a developer) with a compacted view of named data type structures and their relationships, other functionality may be provided by the graphical representations. For example, manipulating fields, named data type structures and related information may be more efficiently executed by using the graphical representations.

Referring to FIG. 9b, graphical representations may be adjusted and manipulated to efficiently construct, reconstruct, etc. the files data type structures, files, etc. For example, representations of data structures may be grouped differently to define new files. As illustrated, the graphical representations of files (e.g., DML file representations 900 and 902) have been removed allowing a developer to adjust the grouping of the graphical representations of the data type structures 904, 906, 908, 910. Along with allowing the data type structures to be reorganized for storage in same or different file or files, such manipulations may improve relationships among the files and reduce the occurrence of unnecessary "include" statements. Just to demonstrate with the illustrated example, the graphical representation 906 (for the data type structure "business_t") may be grouped with the graphical representation 910 (for the data type structure "sales_prospects_t") for more efficient operation. Once grouped, the graphical representations may be initiate file creation for storing the newly grouped data type structures. Along using such operations for grouping and manipulating data type structures, file level operations may also be performed. For example, graphical representations of files (e.g., representations 900, 902) may be manipulated for

combining, removing, appending, etc. the content (e.g., data type structures) of the files.

Referring to FIG. 10, a user interface 1000 is illustrated that provides an editor 1002 for manipulating graphical representations of fields, named data type structures and other types of programming attributes. The editor 1002 includes a window 1004 that presents the graphical representations of the fields, named data type structures and related information (e.g., arrowed lines to represent data type relationships). The editor 1002 also includes a palette 1006 that allows a user (e.g., a developer) to select from a variety of fields, named data type structures, etc. for inclusion in the applications being developed. For example, as represented in the figure by bold, arrowed line 1008, a pointing device may be used to select and insert (e.g., drag and drop) a named data type structure into the window 1004 for project development. Similarly, the selection and insertion operation may be reversed such that a named data type structure is selected from the window 1004 (after being developed) and inserted into the palette 1006. Selection and insertion operations may also be executed solely within the window 1004 or the palette 1006. For example, operations (e.g., drag and drop operations) may be executed in the palette 1006 to create, edit, etc. one or more fields, named data type structures, etc. Similarly, operations (e.g., select, insert, delete, append, etc.) may be initiated by a user in the window 1004 for adjusting fields, named data type structures, etc. Other types of manipulation operations may also be executed, for example, relationships between fields, named data type structures, files (e.g., DML files), etc. may be graphically manipulated.

Referring to FIG. 11, operations for manipulating fields, named data type structures, files, etc. and relationships may be graphically initiated by a user (e.g., a developer). For example, arrowed lines may be manipulated (e.g., deleted, added, moved, etc.) for adjusting relationships among fields and named data type structures. In this illustrated example, two lines 1100 and 1102 are deleted (as represented by graphical symbol "x" being respectively positioned on each line through a user's pointing device). Based upon the relationship being severed between the "project.dml" file and the "CRM.dml" file due to deleting the lines, the "CRM.dml" no longer needs access to the contents of the "project.dml" file. As such, the instruction within the "CRM.dml" file for accessing the "project.dml" file (e.g., "include 'project.dml'") is removed from the "CRM.dml" file (as represented by the dashed-line box 1104).

Correspondingly, the dashed line 1106 that represents this relationship between the "CRM.dml" file and the "project.dml" file may be similarly removed from the graphical representations of the files. Alternatively, when such relationships between files are established or re-established (e.g., by connecting the two files with lines 1100 and 1102), the instruction (e.g., "include 'project.dml'") may be inserted or re-inserted into the appropriate file (e.g., "CRM.dml") and a graphical representation (e.g., the dashed line 1106) may again be presented to illustrate the relationship.

Referring to FIG. 12, as the amount of named fields, named data type structures, files, etc. grow, one or more techniques may be implemented to assist the user (e.g., the developer) to navigate among the potential fields, named data type structures, files, etc. that may be selected for use during application development. For example, one or more graphical representations may present a selectable list of files that include useable fields and named data type structures. In some arrangements, hierarchical listings may be used to assist the user in navigating the files, named data type structures, etc. As illustrated in the figure, a pane 1200 is included in a user interface 1202 that allows the user to navigate among a listing of different packages (e.g., XML processing data types, lookup data types, meta programming data types, date/time data types, metadata types, etc.). In one arrangement, once a selection is made, artifacts defined in the selected package, such as named data type structures and functions may be displayed in a graphical representation located on the right-hand side of the user interface 1202 and manipulated (e.g., selected, navigated, dragged-and-dropped onto the pane 1200) as needed.

Along with organizing, manipulating, etc. fields, named data type structures, files for application development, other types of programming attributes may similarly be graphically represented for assisting developers. For example, functions, variables, etc. used by applications may similarly become unwieldy as more and more such programming attributes are created and stored in libraries for later retrieval and reuse. Other techniques may also be implemented for assisting developers in identifying and selecting appropriate programming attributes. For example, algorithms for logically distributing programming attributes among files (referred to as clustering algorithms) may be used, for example, to organize fields, named data type structures, functions, etc. Through such organizing techniques the amount of instructions (e.g., "include" statements) may be reduced along with redundant use of named data type structures.

Referring to FIG. 13, a flowchart 1300 represents operations of a procedure for graphically representing programming attributes such as named data type structures used in application development. The operations are typically executed by a single computing device (e.g., providing a development environment); however, operations may be executed by multiple computing devices. Along with being executed at a single site, operation execution may be distributed among two or more locations.

Operations may include defining 1302 a data structure representing a hierarchy of one or more programming attributes for developing an application. The graphical representation of the files are removable to allow for manipulating the graphical representations of the data structure to define one or more data structure groups and to create one or more new files for storing the one or more data structure groups. For example, a hierarchy of subrecords, records, fields, named data type structures, etc. may be used to define a data structure. The data structure is defined to be included in a single file (for storage), but in some arrangements the data structure may be included in multiple files (e.g., for storing and later retrieval). Operations also include producing 1304 a visual diagram including a graphical representation of the data structure and a graphical representation the file storing the data structure. The visual diagram also includes a graphical representation of a relationship between the data structure and another data structure and a graphical representation of a relationship between the file storing the data structure and another file storing the other data structure. For example, as shown in FIG. 7, two named data type structures (e.g., "business_t" and "class_t") are graphically represented in a visual diagram that represents the named data type structures included in a file "project.dml". A third data type structure is also presented (e.g., "person_t") that includes a hierarchy of fields that are used by both of the other named data type structures (e.g., "business_t" and "class_t") as indicated by the graphical lines 706, 708 also represented in the visual diagram. Along with illustrating the contents of each data structure, relationships among the data structures are graphically represented, for example, to allow a viewer (e.g., a developer) to relatively quickly ascertain the lineage of fields, named data type structures, etc. and their relationships.

The approach for graphically representing computational artifact described above can be implemented using software for execution on a computer. For instance, the software forms procedures in one or more computer programs that execute on one

or more programmed or programmable computer systems (which may be of various architectures such as distributed, client/server, or grid) each including at least one processor, at least one data storage system (including volatile and non-volatile memory and/or storage elements), at least one input device or port, and at least one output device or port. The software may form one or more modules of a larger program, for example, that provides other services related to the design and configuration of dataflow graphs. The nodes and elements of the graph can be implemented as data structures stored in a computer readable medium or other organized data conforming to a data model stored in a data repository.

The software may be provided on a storage medium, such as a CD-ROM, readable by a general or special purpose programmable computer, or delivered (encoded in a propagated signal) over a communication medium of a network to a storage medium of the computer where it is executed. All of the functions may be performed on a special purpose computer, or using special-purpose hardware, such as coprocessors. The software may be implemented in a distributed manner in which different parts of the computation specified by the software are performed by different computers. Each such computer program is preferably stored on or downloaded to a storage media or device (e.g., solid state memory or media, or magnetic or optical media) readable by a general or special purpose programmable computer, for configuring and operating the computer when the storage media or device is read by the computer system to perform the procedures described herein. The inventive system may also be considered to be implemented as a computer-readable storage medium, configured with a computer program, where the storage medium so configured causes a computer system to operate in a specific and predefined manner to perform the functions described herein.

A number of embodiments of the invention have been described. Nevertheless, it will be understood that various modifications may be made without departing from the spirit and scope of the invention. For example, some of the steps described above may be order independent, and thus can be performed in an order different from that described.

It is to be understood that the foregoing description is intended to illustrate and not to limit the scope of the invention, which is defined by the scope of the appended claims. For example, a number of the function steps described above may be

performed in a different order without substantially affecting overall processing. Other embodiments are within the scope of the following claims.

What is claimed is:

1. A method for representing information, including:

defining a data structure representing a hierarchy of at least one programming attribute for developing an application, wherein the data structure is stored in a file to allow the data structure to be used by other data structures stored in other files; and

producing a visual diagram including a graphical representation of the data structure and a graphical representation of the file storing the data structure, the visual diagram also including a graphical representation of a relationship between the data structure and another data structure and a graphical representation of a relationship between the file storing the data structure and another file storing the other data structure.

2. The method of claim 1, wherein the graphical representation of the files are removable to allow for manipulating the graphical representations of the data structure to define one or more data structure groups and to create one or more new files for storing the one or more data structure groups.

3. The method of claim 1 further including:

manipulating the defined data structure to adjust the at least one programming attribute.

4. The method of claim 1 further including:

manipulating the defined data structure for inserting the at least one programming attribute into a file.

5. The method of claim 4 wherein a statement is inserted with the at least one programming attribute into the file.

6. The method of claim 4 wherein manipulating the data structure includes a drag and drop operation.

7. The method of claim 3, wherein manipulating the defined data structure includes at least one of adding, deleting and editing content of the hierarchy of the programming attribute.

8. The method of claim 1, wherein the programming attribute is a named data type.

9. The method of claim 1, wherein the programming attribute is a function.

10. The method of claim 1, wherein the relationship between the data structure and the other data structure represents lineage of the programming attribute.

11. A computer-readable storage medium storing a computer program for representing information, the computer program including instructions for causing a computing system to:

define a data structure representing a hierarchy of at least one programming attribute for developing an application, wherein the data structure is stored in a file to allow the data structure to be used by other data structures stored in other files; and produce a visual diagram including a graphical representation of the data structure and a graphical representation of the file storing the data structure, the visual diagram also including a graphical representation of a relationship between the data structure and another data structure and a graphical representation of a relationship between the file storing the data structure and another file storing the other data structure.

12. A computing system for representing information, the computing system including:

at least one processor configured to process information, the processing including defining a data structure representing a hierarchy of at least one programming attribute for developing an application, wherein the data structure is stored in a file to allow the data structure to be used by other data structures stored in other files, and

producing a visual diagram including a graphical representation of the data structure and a graphical representation of the file storing the data structure, the visual diagram also including a graphical representation of a relationship between the data structure and another data structure and a graphical representation of a relationship between the file storing the data structure and another file storing the other data structure; and

an output device for presenting the visual diagram that includes the graphical representations of the data structure and file and the graphical representations of the relationship of the data structures and the relationship of the files.

13. A computing system for representing information, the computing system including:

means for processing including

defining a data structure representing a hierarchy of at least one programming attribute for developing an application, wherein the data structure is stored in a file to allow the data structure to be used by other data structures stored in other files, and

producing a visual diagram including a graphical representation of the data structure and a graphical representation of a relationship between the data structure and another data structure; and

means for presenting the visual diagram that includes the graphical representations of the data structure and file and the graphical representations of the relationship of the data structures and the relationship of the files.

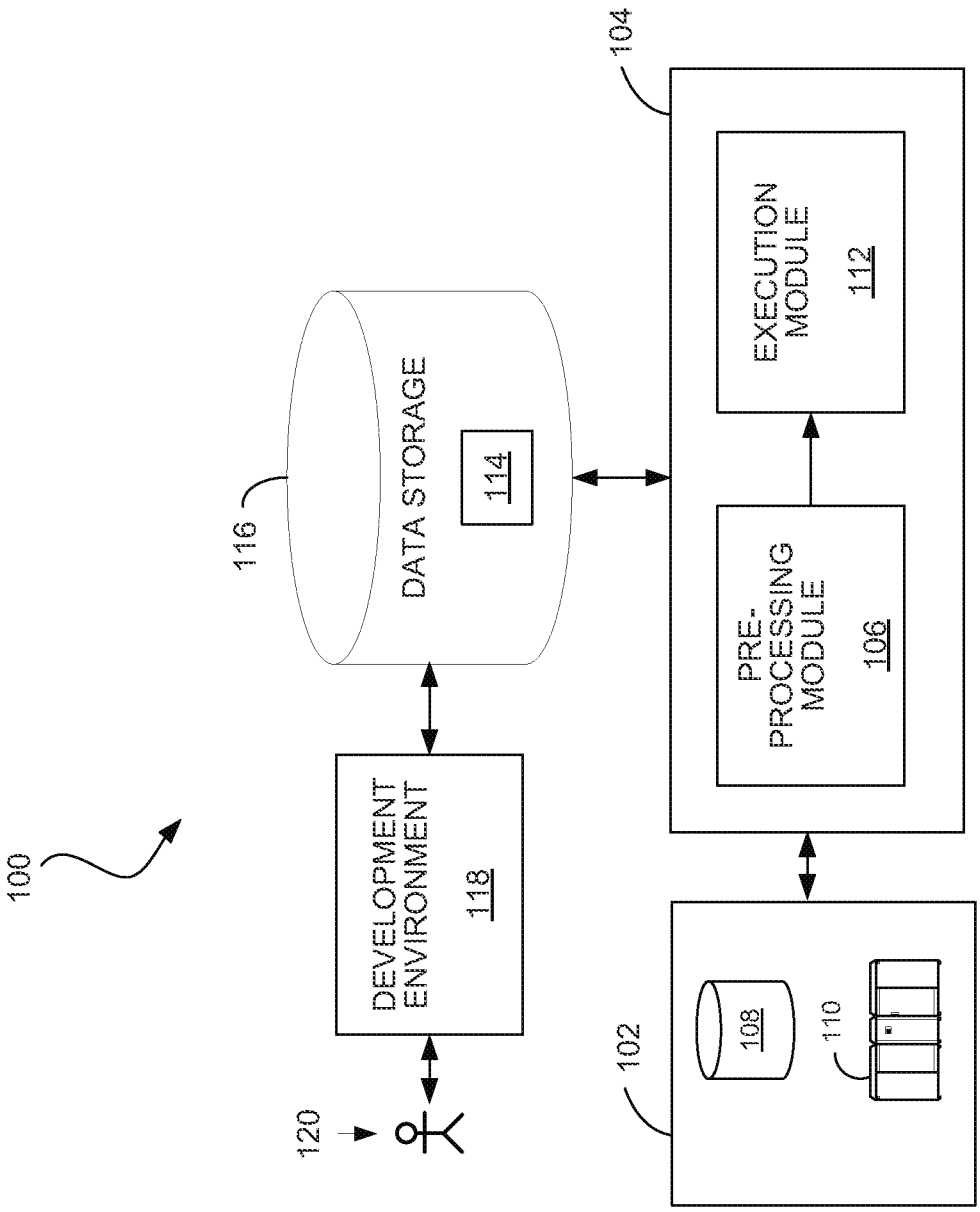
100
120
DEVELOPMENT ENVIRONMENT
118
116
DATA STORAGE
114
104
EXECUTION MODULE
112
PRE-PROCESSING MODULE
106
102
108
110

FIG. 1

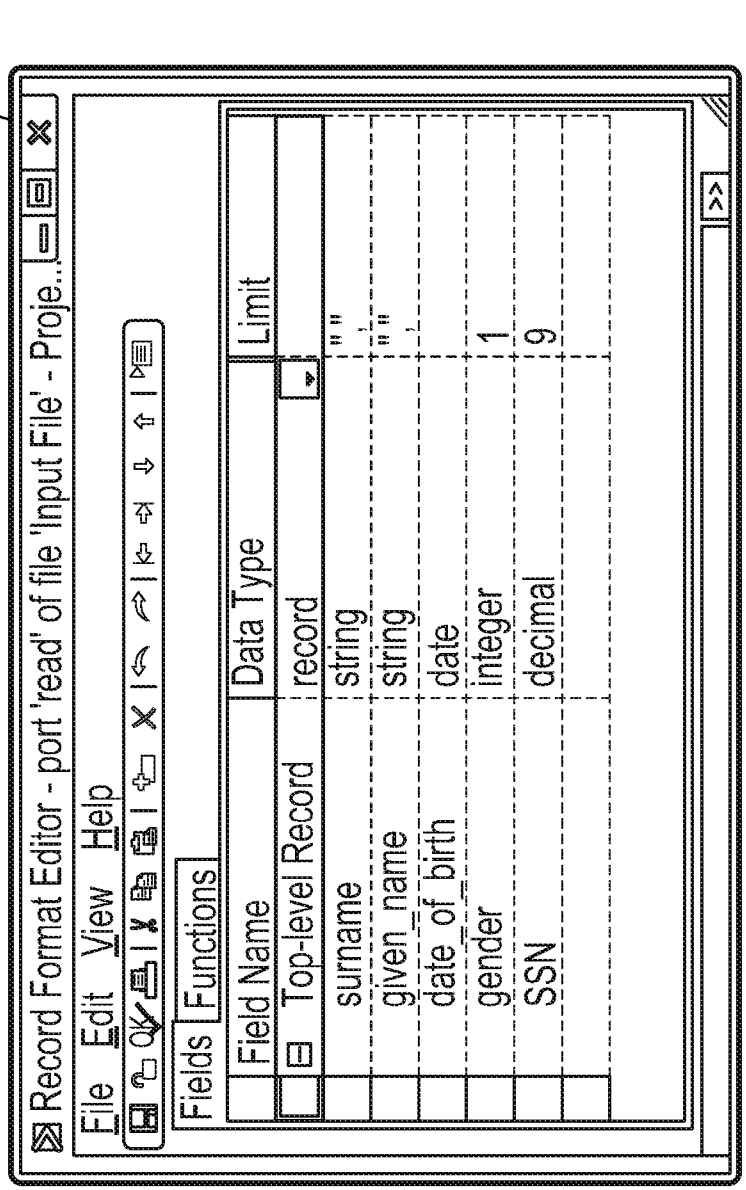
200
Record Format Editor - port 'read' of file 'Input File' - Proje...
File Edit View Help
Fields Functions
Field Name
Data Type
Limit
Top-level Record
record
surname
string
given_name
string
date_of_birth
date
gender
integer
1
SSN
decimal
9

FIG. 2

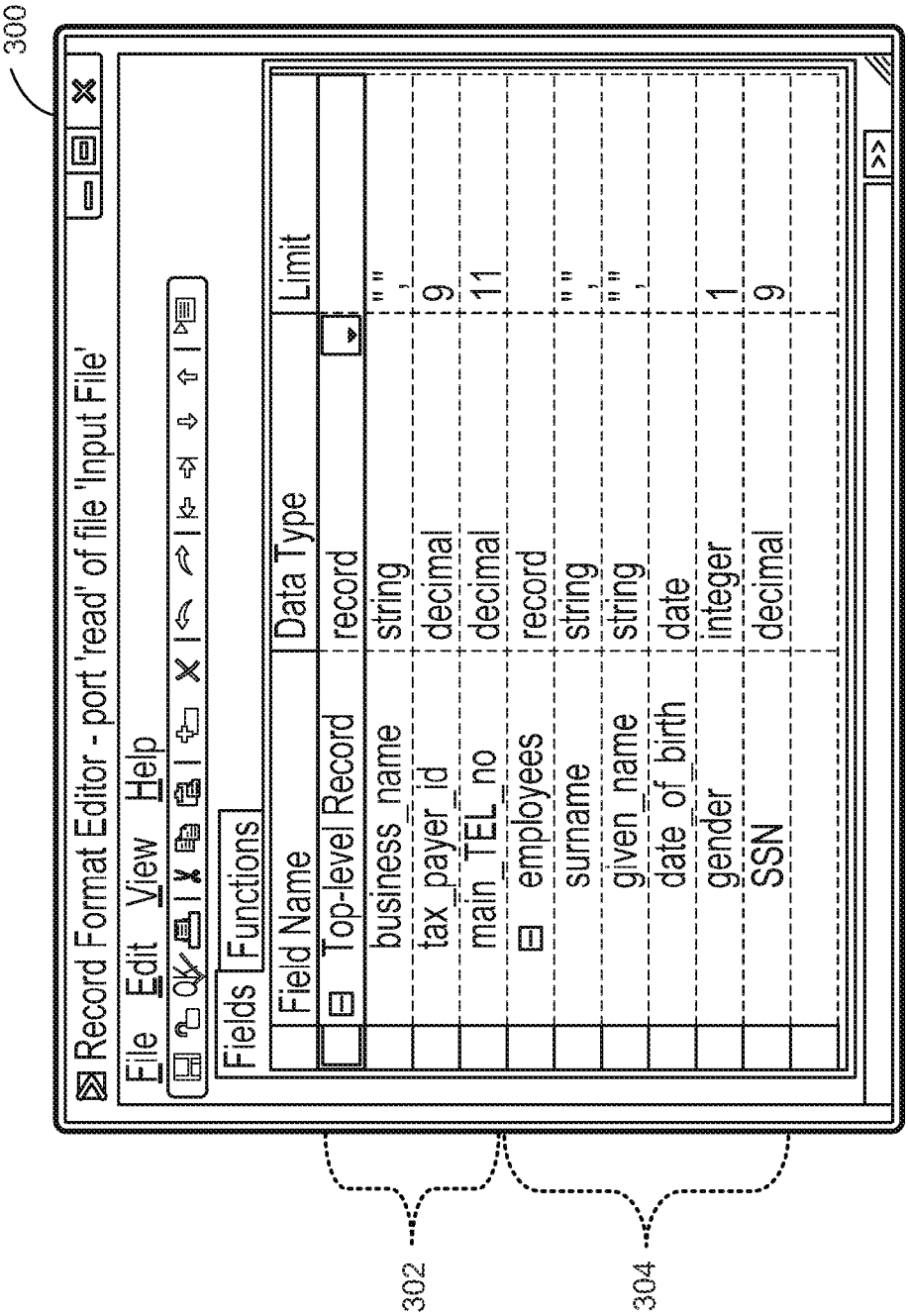
300
Record Format Editor - port 'read' of file 'Input File'
File Edit View Help
Fields Functions
Field Name
Data Type
Limit
Top-level Record
record
business_name
string
""
tax_payer_id
decimal
9
main_TEL_no
decimal
11
employees
record
surname
string
""
given_name
string
""
date_of_birth
date
gender
integer
1
SSN
decimal
9
302
304

FIG. 3

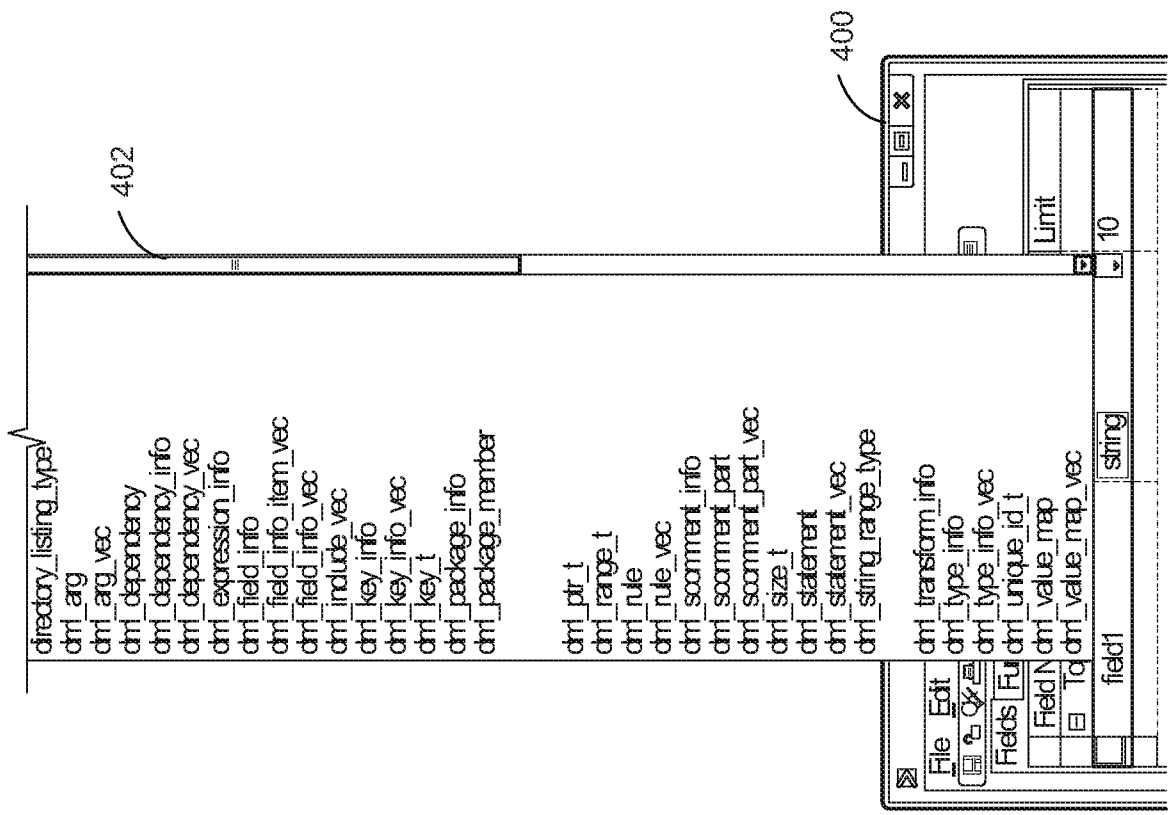
400
402
directory_listing_type
dm_arg
dm_arg_vec
dm_dependency
dm_dependency_info
dm_dependency_vec
dm_expression_info
dm_field_info
dm_field_info_item_vec
dm_field_info_vec
dm_include_vec
dm_key_info
dm_key_info_vec
dm_key_t
dm_package_info
dm_package_member
dm_ptr_t
dm_range_t
dm_rule
dm_rule_vec
dm_scomment_info
dm_scomment_part
dm_scomment_part_vec
dm_size_t
dm_statement
dm_statement_vec
dm_string_range_type
dm_transform_info
dm_type_info
dm_type_info_vec
dm_unique_id_t
dm_value_map
dm_value_map_vec
File Edit
Fields Fu
Field N
Tq
field1
string
Limit
10

FIG. 4

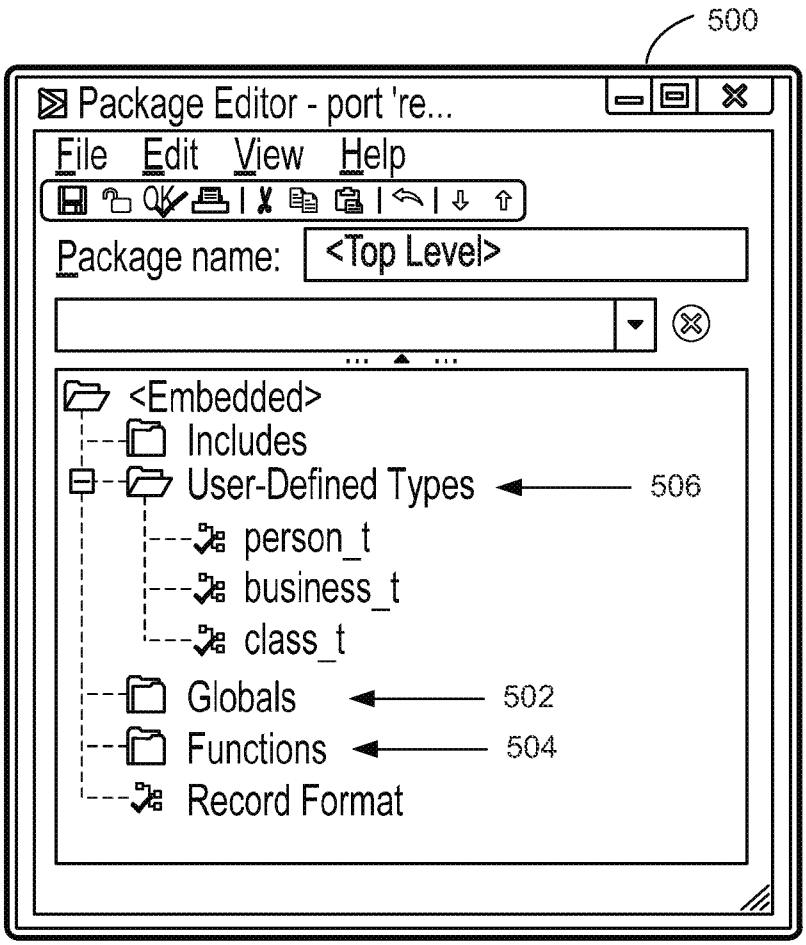
500
Package Editor - port 're...
File Edit View Help
Package name: <Top Level>
<Embedded>
Includes
User-Defined Types
506
person_t
business_t
class_t
Globals
502
Functions
504
Record Format

FIG. 5

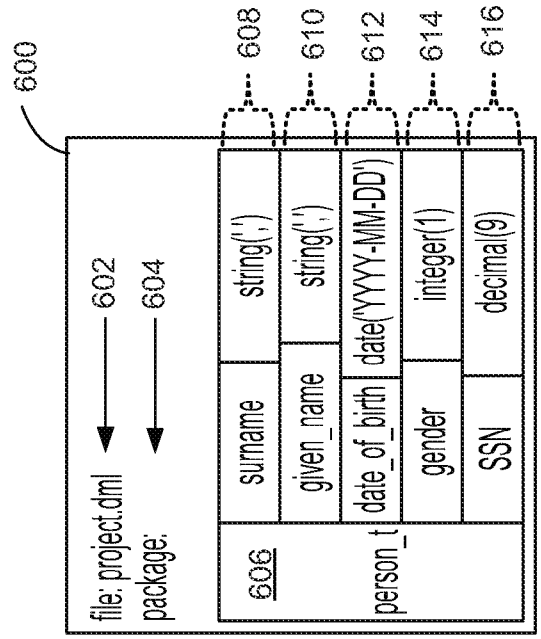
600
608
610
612
614
616
file: project.dml
package:
602
604
606
person_t
surname
given_name
date_of_birth
gender
SSN
string(',')
string(',')
date('YYYY-MM-DD')
integer(1)
decimal(9)

FIG. 6

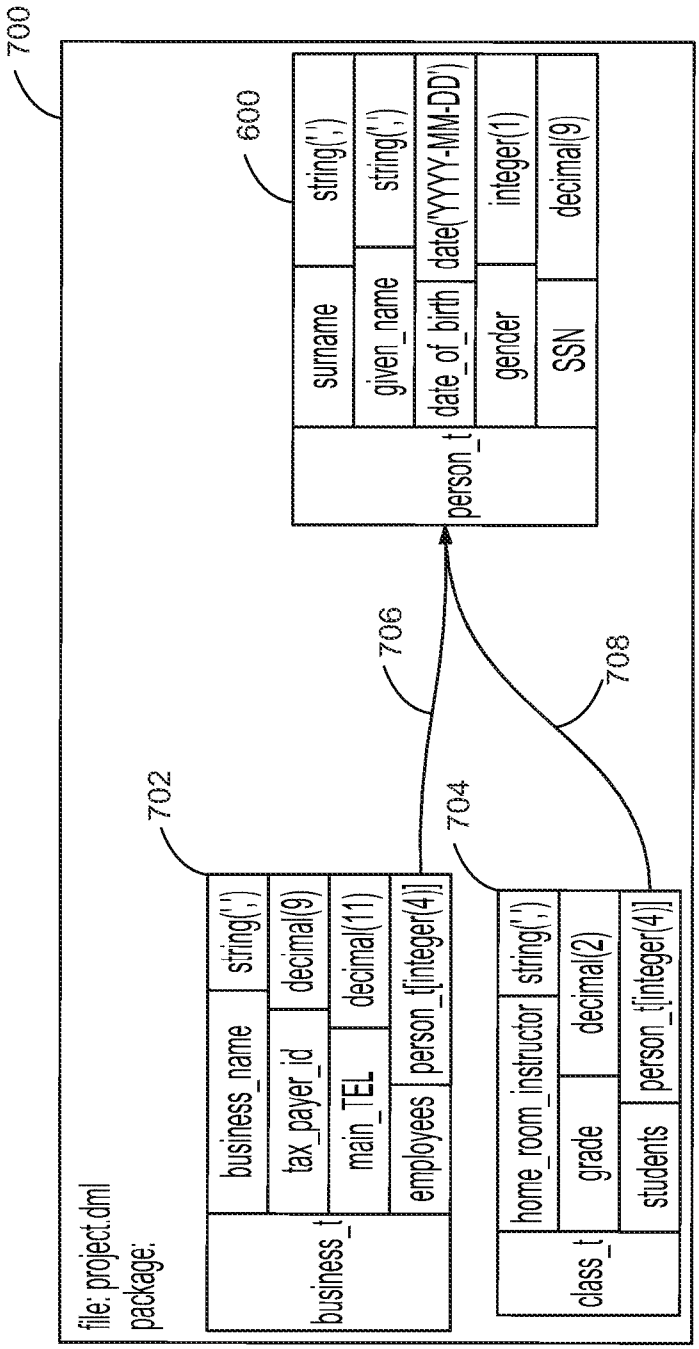
700
file: project.dml
package:
business_t
business_name
string(')
tax_payer_id
decimal(9)
main_TEL
decimal(11)
employees
person_t[integer(4)]
class_t
home_room_instructor
string(')
grade
decimal(2)
students
person_t[integer(4)]
person_t
surname
string(')
given_name
string(')
date_of_birth
date('YYYY-MM-DD')
gender
integer(1)
SSN
decimal(9)
600
702
704
706
708

FIG. 7

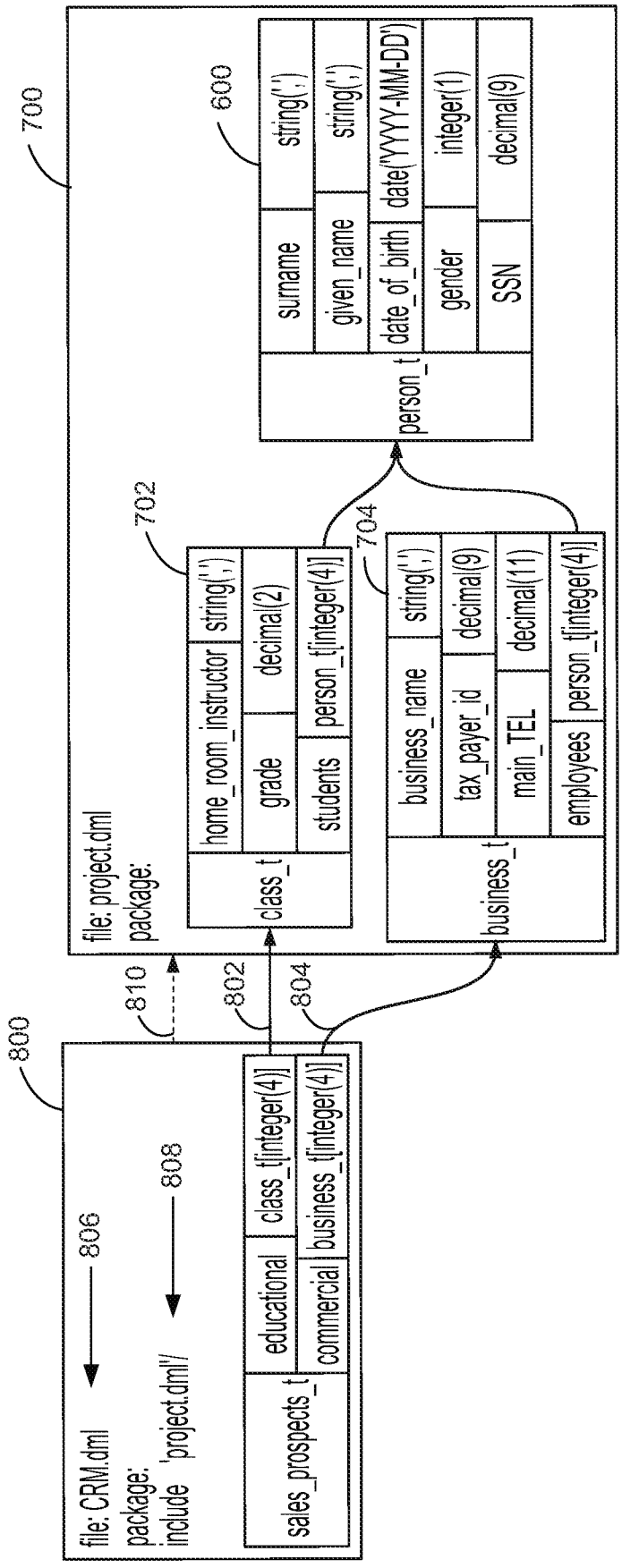
800
file: CRM.dml
package:
include 'project.dml'//
806
808
sales_prospects_t
educational
commercial
class_t[integer(4)]
business_t[integer(4)]
810
802
804
700
file: project.dml
package:
class_t
home_room_instructor
string(')
grade
decimal(2)
students
person_t[integer(4)]
702
business_t
business_name
string(')
tax_payer_id
decimal(9)
main_TEL
decimal(11)
employees
person_t[integer(4)]
704
600
person_t
surname
string(')
given_name
string(')
date_of_birth
date('YYYY-MM-DD')
gender
integer(1)
SSN
decimal(9)

FIG. 8

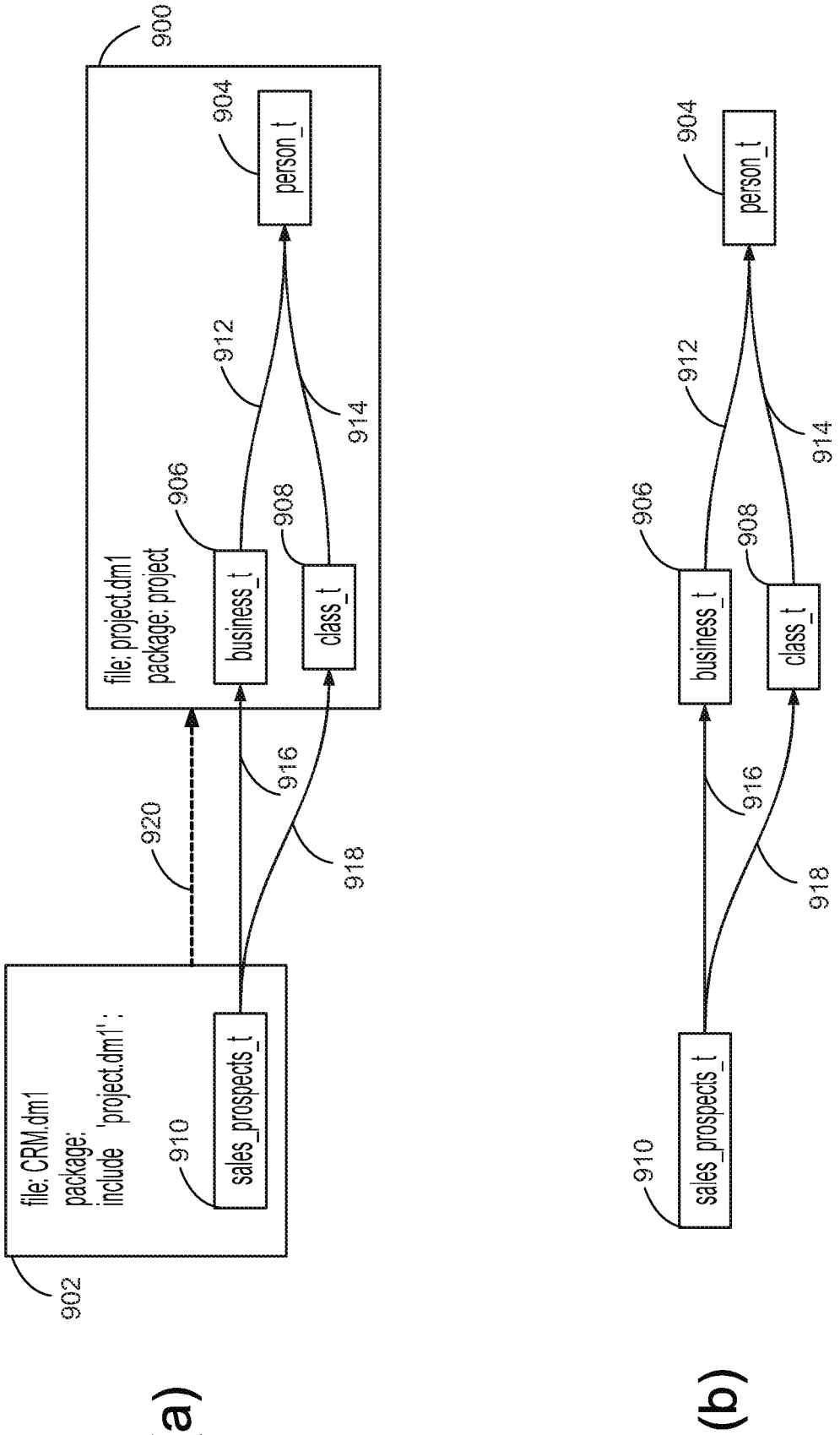
(a)
file: CRM.dm1
package:
include 'project.dm1';
sales_prospects_t
910
902
920
file: project.dm1
package: project
business_t
906
class_t
908
person_t
904
912
914
916
918
900
(b)
sales_prospects_t
910
business_t
906
class_t
908
person_t
904
912
914
916
918

FIG. 9

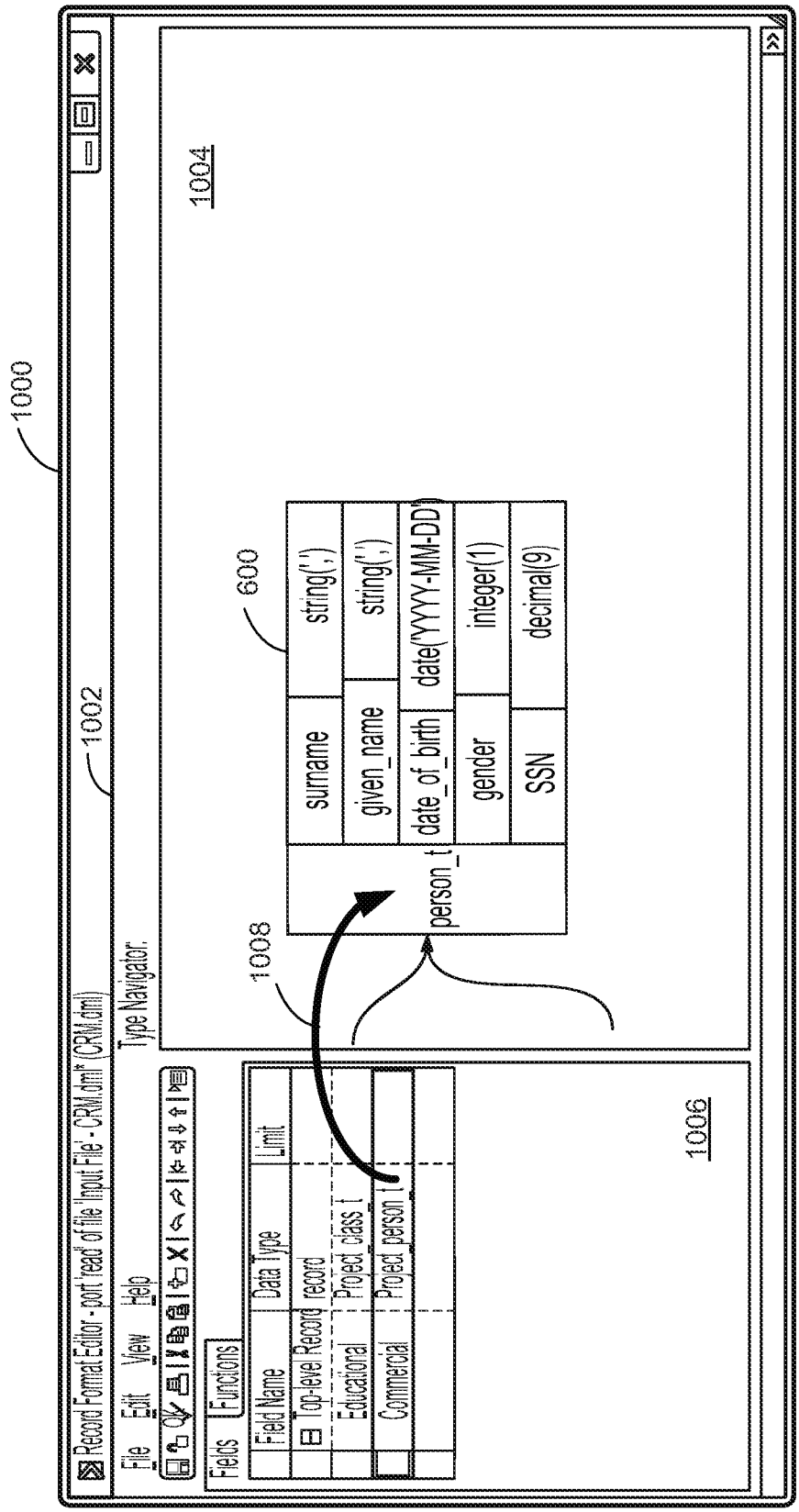
1000
1002
1004
1006
1008
600
Record Format Editor - port 'read' of file 'Input File' - CRM.dml* (CRM.dml)
File Edit View Help
Type Navigator
Fields Functions
Field Name Data Type Limit
Top-level Record record
Educational Project_class_t
Commercial Project_person_t
person_t
surname string(',')
given_name string(',')
date_of_birth date('YYYY-MM-DD')
gender integer(1)
SSN decimal(9)

FIG. 10

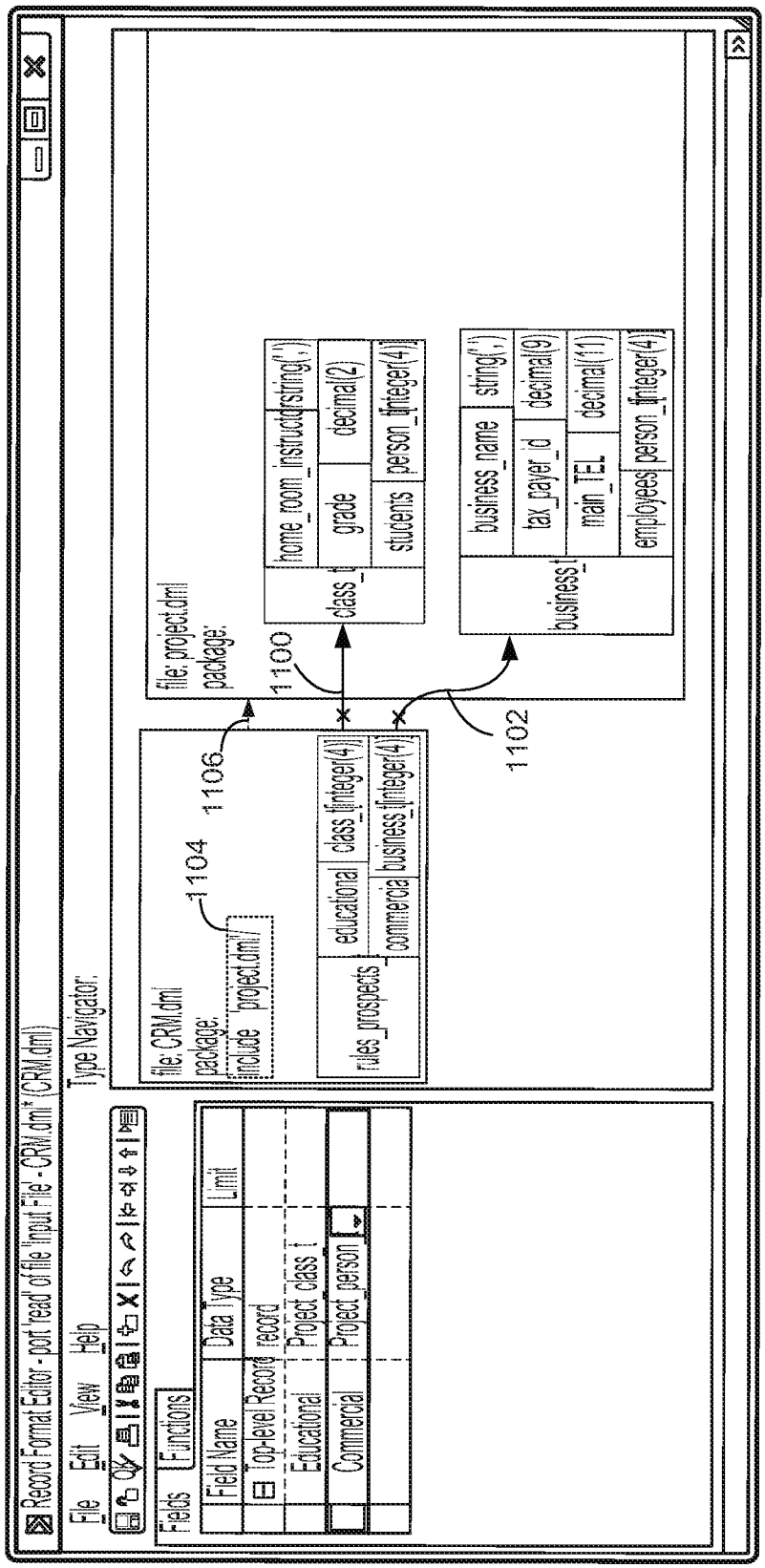
Record Format Editor - port 'read' of file 'Input File' - CRM.dml* (CRM.dml)
File Edit View Help
Fields Functions
Field Name Data Type Limit
Top-level Record record
Educational Project class 1
Commercial Project_person
Type Navigator:
file: CRM.dml
package:
include "project.dml"
rules_prospects
educational class_t(integer(4))
commercial business_t(integer(4))
file: project.dml
package:
class_t
home_room_instructdrstring(1")
grade decimal(2)
students person_t(integer(4))
business_t
business_name string(1")
tax_payer_id decimal(9)
main_TEL decimal(11)
employees person_t(integer(4))
1100
1102
1104
1106

FIG. 11

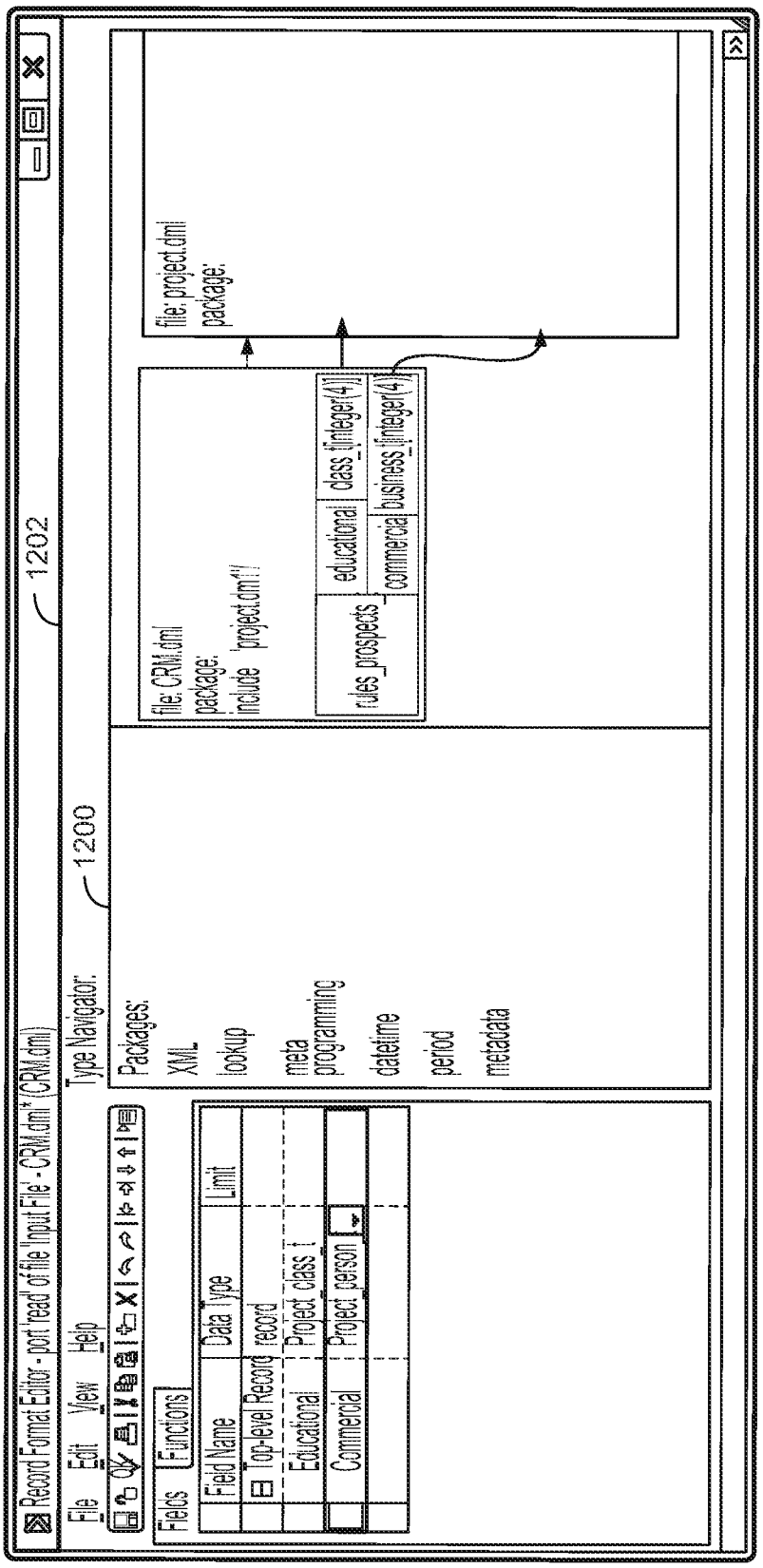
Record Format Editor - port 'read' of file 'Input File' - CRM.dmi * (CRM.dmi)
File Edit View Help
Fields Functions
Field Name Data Type Limit
Top-level Record record
Educational Project class_t
Commercial Project_person
1202
Type Navigator:
1200
Packages:
XML
lookup
meta
programming
datetime
period
metadata
file: CRM.dmi
package:
include project.dmi"
rules_prospects
educational class_t(integer(4))
commercial business t(integer(4))
file: project.dmi
package:

FIG. 12

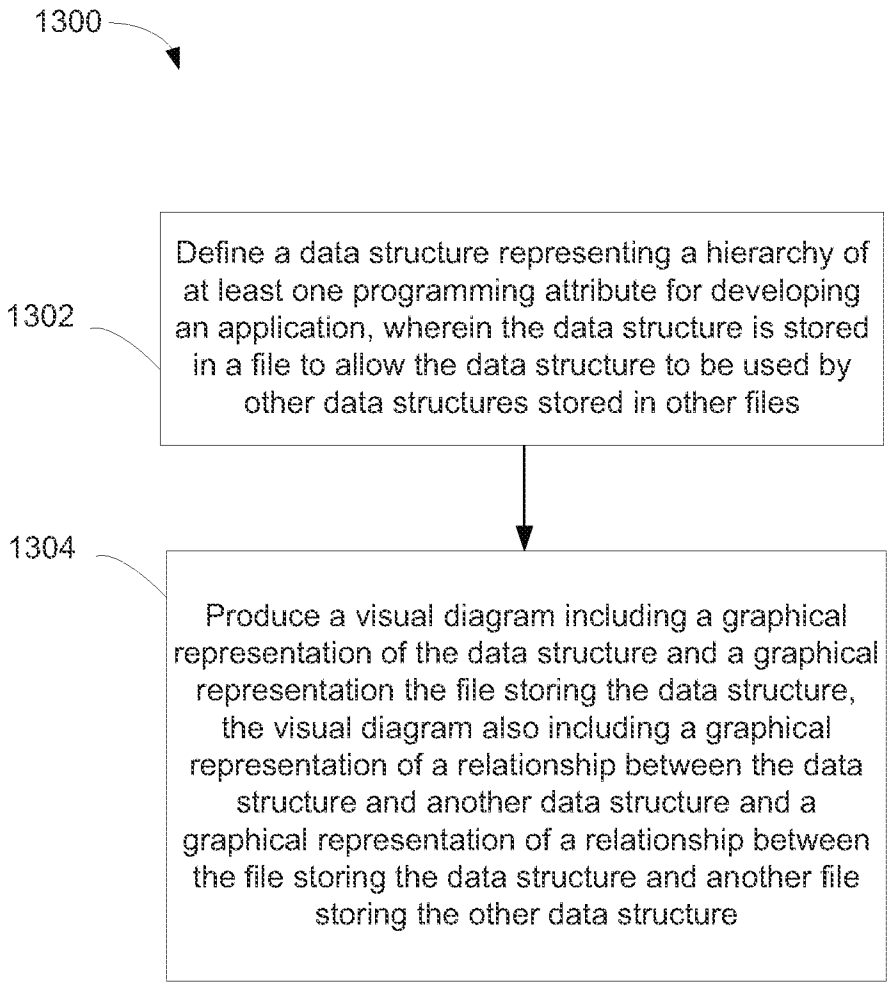
1300
1302
Define a data structure representing a hierarchy of at least one programming attribute for developing an application, wherein the data structure is stored in a file to allow the data structure to be used by other data structures stored in other files
1304
Produce a visual diagram including a graphical representation of the data structure and a graphical representation the file storing the data structure, the visual diagram also including a graphical representation of a relationship between the data structure and another data structure and a graphical representation of a relationship between the file storing the data structure and another file storing the other data structure

FIG. 13

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2013/062369

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F9/44
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 01/82068 A1 (TOGETHERSOFT CORP [US]) 1 November 2001 (2001-11-01) the whole document -----	1-13
X	WO 01/82072 A1 (TOGETHERSOFT CORP [US]) 1 November 2001 (2001-11-01) the whole document -----	1-13

☐ Further documents are listed in the continuation of Box C.

☒ See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
13 February 2014	28/02/2014

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Rackl, Günther

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/US2013/062369

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 0182068 A1	01-11-2001	AU 5371201 A	07-11-2001
		EP 1290552 A1	12-03-2003
		US 2002097253 A1	25-07-2002
		WO 0182068 A1	01-11-2001
WO 0182072 A1	01-11-2001	AU 5910801 A	07-11-2001
		EP 1292887 A1	19-03-2003
		US 2002032900 A1	14-03-2002
		WO 0182072 A1	01-11-2001