



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

(21) International Application Number:
PCT/US2016/029236

(22) International Filing Date:
25 April 2016 (25.04.2016)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant: **HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP** [US/US]; 11445 Compaq Center Drive W., Houston, Texas 77070 (US).

(72) Inventors: **KIRSHENBAUM, Evan R.**; 1501 Page Mill Rd., Palo Alto, California 94304-1100 (US). **SPENCE, Susan Deborah**; 1501 Page Mill Rd., Palo Alto, California 94304-1100 (US).

(74) Agent: **DAS, Manav** et al.; Hewlett Packard Enterprise, 3404 E. Harmony Road Mail Stop 79, Fort Collins, Colorado 80528 (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, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, 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, ST, 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).

Declarations under Rule 4.17:

- as to the identity of the inventor (Rule 4.17(i))
- as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))

Published:

- with international search report (Art. 21(3))

(54) Title: PERSONA OBJECT TO ACCESS DATA

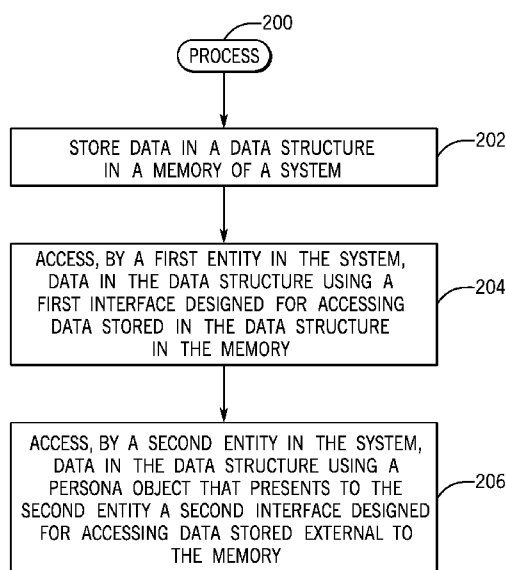


FIG. 2

(57) Abstract: In some examples, a memory stores data in a data structure. A first entity accesses data in the data structure using a first interface designed for accessing data stored in the data structure in the memory of the system. A second entity accesses data in the data structure using a persona object that presents to the second entity a second interface designed for accessing data stored external to the memory of the system.



PERSONA OBJECT TO ACCESS DATA

Background

[0001] Data can be stored in storage devices (such as disk-based storage devices or solid state drives), or alternatively, in memory devices. Memory devices generally exhibit faster access times than storage devices, such that overall system performance can be improved if programs are able to use data or execute instructions stored in memory devices.

Brief Description Of The Drawings

[0002] Some implementations of the present disclosure are described with respect to the following figures.

[0003] Fig. 1 is a block diagram of a system according to some examples.

[0004] Fig. 2 is a flow diagram of a process that uses a persona object according to some examples.

[0005] Figs. 3 and 4 are flow diagrams of processes for retrieving data according to various examples.

[0006] Fig. 5 is a flow diagram of a process of modifying data according to some examples.

[0007] Fig. 6 is a flow diagram of a process that returns a result as a persona object, according to alternative examples.

[0008] Fig. 7 is a block diagram of a file system according to other examples.

[0009] Fig. 8 is a block diagram of an example storage medium storing instructions according to some examples.

Detailed Description

[0010] Systems can include two types of components used for storing data. Data storage devices are those that retain their data when power to the device is removed, and this form of storage can be called “persistent” or “non-volatile” storage. One example type of a data storage device is a magnetic disk drive (other example types of data storage devices are listed further below). Memory devices, by contrast, are those in which it is efficient to access and modify small pieces of data located at arbitrary locations—memory devices that support such access can be referred to as “random access” memory devices. One example type of a memory device is a dynamic random-access memory (DRAM) device (other example memory devices are listed further below). A DRAM device is an example of “volatile” memory device, so data stored in the DRAM device will not survive a loss of power to the DRAM device.

[0011] While these two types allow overlap (i.e. allow there to be components that are simultaneously data storage devices and memory devices), the archetypical components do not overlap. Disk drives are much slower than DRAM devices, and as a result disk drives are not used as random-access devices. Rather care is taken that a disk drive is used in a manner that is both sequential (bytes are read from or written to consecutive locations on the device) and “block-based” (large numbers of bytes are written in a single operation, to amortize the cost of using the device).

[0012] Computer programs access persistent data in three phases (each of which may take place many times). In the first phase, data is read from a data storage device into the memory of the system (e.g. a memory device). The data coming from the data storage device is assumed to be in a format that is optimized for sequential, block-based access (reading and/or writing). This operation can involve a transformation of the sequential data into a data structure (or multiple data structures) in the memory, where a data structure is a particular representation of data that allows for efficient implementation of a set of operations that the program will perform on the data. The data structure that is stored in the memory can be referred to as an “in-memory data structure.” The efficiency is predicated on the

random-access nature of accessing the data at arbitrary locations in the memory (e.g. by dereferencing pointers or references or by indexing into arrays). The representation of data provided by the data structure can be optimized for random access, where “optimized for random access” can mean that the data is in a form where accessing arbitrary elements of the data is made more efficient. The set of operations defines an interface supported by the data structure. Such an interface for a data structure may be called a “data type” or an “abstract data type” supported by the data structure. An example data type is a “list” data type (or interface), which supports finding an element at a particular position in the list, deleting the element at a particular position, inserting an element before a particular position, iterating over elements, and obtaining the number of elements in the list. An example data structure in the C++ language that supports the list interface would be a linked list, where the elements are represented by class objects each containing an element value and a pointer to the next element in the list (or the null pointer to indicate that there are no more such elements). The linked list further includes a pointer to the first element of the list and a number indicating the length of the list.

[0013] In the second phase of data access, the program executes machine-readable instructions to access the in-memory data structure(s) in the memory, where these instructions provide implementations of the interfaces (i.e. the sets of operations in the data types) supported by the data structure(s). During the second phase, if the processor on which the program is running should lose power or if the program should terminate, either intentionally or due to an error or action by an outside entity (e.g. a system administrator), the contents of the data structure(s) in the memory are assumed to be inaccessible to programs that run in the future.

[0014] When the program reaches a state in which the program has identified data that should be made accessible to future programs (or future executions of the same program), the program enters the third phase, during which the in-memory data structure(s) is (are) written to a data storage device (either the same data storage device as was used in the first phase or a different data storage device). In the third phase, a transformation from a random-access-optimized representation of data to a

sequential-access-optimized representation of data can be performed, where the sequential-access-optimized representation of data can better utilize the block-based operations used to write to (and later, to read from) the data storage device.

[0015] In this disclosure, the term “data storage interface” can include a set of operations that are designed for transferring data to or from a storage (which can include a data storage device or multiple data storage devices) to either make the data available persistently or to access persistent data. An example of a data storage interface is a data storage application programming interface (API). A data storage interface is designed to reflect the assumptions of slowness and preference for sequential and block-based access that characterize data storage devices.

Examples of data storage interfaces can include any of the following: a file interface (which accesses data in the form of files from a storage), a directory interface (which accesses data in the form of directories from a storage), a file system interface (which accesses data in the form of file system elements, which include files and directories, from a storage), a database interface (which accesses data in the form of database objects, such as tables, views, etc., from a storage), a key-value store interface (which accesses data stored in a key-value store in a storage), or another type of interface.

[0016] Data storage devices that can be used to support data storage interfaces include magnetic disk drives, tape drives, flash memory drives, optical disk drives (e.g. compact disc (CD), digital versatile disc (DVD), or Blu-ray drives). Alternatively, non-volatile memory devices can also support data storage interfaces. Non-volatile memory devices can include resistive memory devices, memristor memory devices, phase-change memory devices, or spin-transfer torque memory devices. Data storage interfaces may also be implemented using DRAM devices, providing similar behavior without the guarantee of non-volatility. Although they may be implemented to access data stored in a memory device in a given system, data storage interfaces (e.g. data storage APIs), including data storage interfaces presented by persona objects, are designed to access data stored in one or more data storage devices external to the memory of the system. Persona objects are discussed further below.

In particular, data storage interfaces are designed to access persistent data on data storage devices such as magnetic disk drives, in which sequential access to data is far more efficient than random access.

[0017] In this disclosure, the term “in-memory data structure interface” can refer to a set of operations that are designed for efficiently manipulating data, stored in an in-memory data structure, where the operations support accessing arbitrary elements of data or where the operations employ the ability to access arbitrary memory locations for their efficient implementation. An example of an in-memory data structure interface is an in-memory data structure API. Examples of in-memory data structure interfaces include any of the following: a list interface, a tree interface, a map interface, a set interface, an array interface, a record interface, a graph interface, a table interface, and so forth. Memory devices that can be used to support in-memory data structure interfaces include a DRAM device, a static random-access memory (SRAM) device, a read-only memory (ROM) device, an erasable programmable read-only memory (E-PROM) device, and non-volatile memory devices as listed above.

[0018] The recent development of non-volatile memory technologies that are (or will soon be) competitive with DRAM technology in terms of cost, scale, and performance can give rise to a new form of programming in which non-volatile memory devices (collectively referred to as “memory”) are used as both memory devices and data storage devices and in which the three phases described above collapse into a single phase in which the persistent data is already in the memory, can be manipulated there, and, when done, is already persistently stored in the memory (and ready to be used by other programs in the future). This manner of storing and accessing data in a memory can be much more efficient, since the first and third phases of reading and writing the data can be omitted or, at least reduced to simply obtaining a reference to the data and making such a reference available. In such implementations, a representation of data can be provided in the memory, and a system would not have to provide a second representation of data that is

optimized for data storage operations (e.g. one optimized for sequential reads and writes).

[0019] But moving to the foregoing approach to programming is not without cost. There is a large corpus of existing programs in current systems, and the vast majority of these programs were written using data storage interfaces and the read-manipulate-write approach (the three-phase data access approach discussed above). Moreover, many current programmers are trained to write programs in this manner, so having them write programs in the new manner would involve significant retraining cost, even if the resulting programs may be more efficient. Because of this, programs (or parts of programs, such as libraries) using non-volatile memory for data storage can be divided into “old-style” or “legacy” programs and “new-style” programs. Legacy programs either already exist and use data storage interfaces or are newly written to use data storage interfaces. For such legacy programs, a non-volatile memory device may be used in the manner that a data storage device would be used, where data in the non-volatile memory is represented sequentially when at rest (not being accessed by a program), and read operations create copies in data-structure representations either in memory, implemented with non-volatile memory device(s) or volatile memory device(s). New-style programs use in-memory data structure interfaces to manipulate data directly in non-volatile memory. An issue that arises is that legacy and new-style programs cannot easily share data, leaving organizations with the choice between (1) writing all programs in the old style, with its inefficiencies, (2) writing (or rewriting) all programs in the new style and converting existing data in data storage devices to data structures to be stored in memory, which involves rewriting working code and retraining programmers, or (3) allowing programs to be written in either style and maintaining two copies of persistent data, somehow keeping the copies of persistent data synchronized with one another.

[0020] In accordance with some implementations of the present disclosure, a different technique can be used, in which data is stored in a representation (in the form of an in-memory data structure) that is efficient for new-style programs using in-

memory data structure interfaces, but existing or newly-written legacy programs (i.e. new programs written using legacy techniques) can access and modify the same data using data storage interfaces, where such access is efficient and does not involve maintaining a separate copy of the data and where modifications made by new-style programs can be made visible to legacy programs and vice versa. To enable this data sharing, the notion of a “data structure persona object” is introduced, where the data structure persona object is an object in the (volatile or non-volatile) memory of a system that presents a data storage interface and implements the operations of this data storage interface by making calls on an in-memory data structure interface for an in-memory data structure associated with the persona. In some examples, an in-memory data structure may itself support one or multiple data storage interfaces. In such examples, the in-memory data structure may also be a persona object and it may be its own associated in-memory data structure.

[0021] A persona object can refer to an object (which can be in the form of machine-readable instructions and associated information) that acts as an intermediary between a program that is able to request access to data, and an in-memory data structure stored in a memory. A persona object can also be referred to as an adapter. The persona object allows data stored in an in-memory data structure to be presented as data according to a different arrangement expected by a requesting program, where the different arrangement of data can include a file, a directory table, a database table, etc. Thus, by using a persona object, data in an in-memory data structure can pretend to be data according to a different arrangement.

[0022] While a first program is interacting with a given in-memory data structure via a persona object, other programs (or other parts of the first program) can concurrently interact directly with the same in-memory data structure using in-memory data structure interfaces. Programs “concurrently” accessing data in the in-memory data structure can refer to the programs executing instructions to perform operations to access data in the in-memory data structure, where the operations are pending at the same time for processing by a system.

[0023] Although reference is made to new-style or legacy programs accessing data using an in-memory data structure interface or data storage interface, it is noted that other types of entities can also use such interfaces. Such other entities can include hardware devices, for example. Thus, the term “entity” can refer to a program (including machine-readable instructions), an operating system process, a program thread within an operating system process, an operating system thread, a library, a device driver, a hardware device, or any other requester of data. A new-style entity is an entity that uses an in-memory data structure interface to access persistent data stored in an in-memory data structure, while a legacy entity uses a data storage interface to access persistent data. According to some examples, a legacy entity is able to access data in an in-memory data structure using a data storage interface of a persona object. In some examples, a given entity may be a legacy entity with respect to some persistent data and a new-style entity with respect to other persistent data. In some examples, an entity may include as sub-entities both new-style and legacy entities (e.g. a new-style program that includes invocation of functions in a legacy library or vice versa).

[0024] Fig. 1 is a block diagram of an example system 100 (e.g. a computer node or an arrangement of multiple computer nodes) that includes a memory 102, which can be a non-volatile memory, in some examples. It is noted that the memory 102 can be implemented with one memory device (or a portion of the memory device), or can be implemented with multiple memory devices (or portions thereof), where the multiple memory devices can be distributed across a large number of computer nodes in some examples.

[0025] The memory 102 stores data 104 in a data structure 106 (referred to as an “in-memory data structure”) that is of (or supports) a respective data type (also referred to as a “data structure type”). As discussed above, a data type describes a logical entity by specifying an interface that comprises a set of operations that may be performed on a data structure that implements that data type. Examples of data types include a primitive data type (such as a real number, an integer number, a Boolean value, etc.), a list data type (that enable access to data as lists), a graph

data type (that enable access to data as graphs), a set data type (that enable access to data as sets), an array data type (that enables access to data as arrays), a string data type (that enables access to data as strings), a stack data type (that enables access to data as stacks), a queue data type (that enables access to data as queues), a vector data type (that enables access to data as vectors), a matrix data type (that enables access to data as matrices), a tree data type (that enables access to data as trees), and so forth. Further examples of data structure types can include user-defined data types (also called “records”, “structs”, or “classes”), such as those defined by a programmer (or programmers) or other users or other entities (programs and/or hardware devices) that enable access to data as a collection of named fields, each associated with a respective data type.

[0026] A given data structure 106 may support multiple data types. A given data structure 106 is an instance of a data structure implementation type, which specifies the way that the data 104 contained in the given data structure is represented in the memory 102 and also specifies the manner (e.g. by means of machine-readable instructions) that the in-memory representation is accessed to implement the operations of the data type or data types supported by the data structure 106. A given data type may be supported by data structures that are instances of multiple data structure implementation types. For example, a list data type may be implemented by instances of a linked list implementation type, which represents a list in the memory 102 as individual nodes, where each node contains a value and a pointer to the next node in the list. The list data type may also be implemented by instances of an array list implementation type, which represents a list in the memory 102 as an array of elements.

[0027] Although just one in-memory data structure 106 is shown in Fig. 1, it is noted that in other examples, the memory 102 can contain multiple in-memory data structures according to one or multiple implementation types and supporting data structure types. Multiple in-memory data structures can include multiple instances of a specific data structure type, or can include instances of respective different data structure types.

[0028] The system 100 also includes a first interface 108 (e.g. a first API or data type) designed to access data in the in-memory data structure 106. The first interface 108 is an in-memory data structure interface. The first interface 108 includes an operation 110 (or multiple operations) that can be invoked to access data in the in-memory data structure 106. The operation 110 is in the form of machine-readable instructions (e.g. routines, methods, or functions). A first entity 112 (program or hardware device) in the system 100 (or external of the system 100) can use the first interface 108 to access data in the in-memory data structure 106. More specifically, the first entity 112 can invoke the operation 110 to perform the access of data in the in-memory data structure 106.

[0029] If there are multiple in-memory data structures of different data structure types stored in the memory 102, then there can be respective multiple in-memory data structure interfaces with corresponding operations to access data of the respective different in-memory data structures.

[0030] The system 100 further includes a persona object 114 that presents, to a second entity 118 (a legacy entity), a second interface 116 designed for accessing data stored external of the memory 102, such as in an external storage that is not shown in Fig. 1 (and may not actually exist in the system 100). The second interface 116 includes an operation 117 (or multiple operations) that can be invoked by the second entity 118 to cause the persona object 114 to access data in the in-memory data structure 106. The result of the invocation of the operation 117 is returned to the second entity 118 in a form expected by the second entity 118, such as in the form of a string representing a line (e.g. a record encoded as comma-separated values) read from a file, a file enumerated in a directory, a collection of values from a row in a database table, a value associated with a key in a key-value store, and any other data arrangement that is expected to be stored external of the memory 102.

[0031] Although just one persona object 114 is shown in Fig. 1, it is noted that in other examples, there can be multiple persona objects in the system 100, where the multiple persona objects can be according to different persona types. The multiple persona objects according to the different persona types can present different

interfaces to respective legacy entities that expect different interfaces (e.g. a database interface, a file system interface, etc.). More generally, if the system 100 includes multiple in-memory data structures, then each in-memory data structure can be associated with one or multiple persona objects and each persona object can be associated with one or multiple in-memory data structures.

[0032] In accordance with some examples, the persona object 114 avoids maintaining multiple copies of the same data that is in the in-memory data structure 106 within the system 100. In other words, the system 100 does not have to maintain multiple representations of the same data contained in the in-memory data structure 106 in order to support both the first interface 108 and the second interface 116. Rather, there is a single internal representation of the data 104 of the in-memory data structure 106, and one or multiple interpretations of that data through the use of one or multiple respective persona objects.

[0033] The persona object 114 is able to translate a request submitted to the second interface 116 (such as a call to the operation 117 in the second interface 116) into a request that is able to access the in-memory data structure 106 (such as one or multiple calls to operation 110 on the first interface 108). Effectively, the presence of the persona object 114 allows the data 104 stored in the in-memory data structure 106 to be used as data stored in a storage or another memory external of the memory 102 and which is accessed by an entity using a legacy interface.

[0034] In addition, by using the persona object 114 according to some implementations of the present disclosure, no conversion of data between the format of the in-memory data structure 106 and a different format expected by a legacy entity (e.g. the second entity 118) has to be performed.

[0035] By using persona objects according to some examples, legacy entities do not have to be modified to support an in-memory data structure interface (e.g. the first interface 108); rather, the legacy entities can access legacy interfaces (e.g. the second interface 116) provided by the respective persona objects.

[0036] Also, any change of the data 104 in the in-memory data structure 106 can be made visible to a legacy entity in a real-time manner (i.e. when the data 104 is changed, any subsequent request to access the data 104 from a legacy entity via the persona object 114 can reflect the change). Similarly, as a legacy entity causes a change to the data 104 in the in-memory data structure 106 (such as by issuing a modification request), the change in the data 104 can also be made visible in a real-time manner to an entity (e.g. 112) that accesses the data 104 using an in-memory data structure interface (e.g. 108).

[0037] In some examples, a single data structure 106 may be associated with multiple persona objects 114 that each present the same interface 116 but do so by means of different implementations. For example, the data structure 106 may support a first interface 108 that presents the data 104 as a list of records each containing a name (as a string) and a score (as an integer), and each of the persona objects 114 may present as the second interface 116 a file interface that contains an operation 117 for reading the next line of the file and returning an instance of a string data type representing that line. The persona objects 114 may implement the read operation 117 by identifying a record in the list 106 and returning a string that describes it, but the manner of determining the string may be different depending on the type of the file each persona object 114 is pretending to be and the expectations of the various legacy entities 118 using them. A first legacy entity 118 may desire a file whose lines contain comma-separated values (a CSV file). A persona object 114 pretending to be such a file and desiring to return a string corresponding to a record containing a name of "Fred" and a score of 12 might return "Fred,12" (i.e. a string containing the characters "F", "r", "e", "d", ",", "1", and "2"), with the expectation that the invoking entity 118 will split this at the comma into two substrings "Fred" and "12" and will parse the string "12" into the number twelve. A second legacy entity 118 may desire a file whose lines contain key-value pairs, with the keys separated from the values by equals (=) signs. A proxy 114 pretending to be such a file and desiring to return a string corresponding to the same record might return "score=12; name=Fred".

[0038] Fig. 2 is a flow diagram of an example process 200 that can be performed by a system, such as the system 100. The process 200 includes storing (at 202) data in a data structure (e.g. 106) in a memory (e.g. 102) of the system. The process 200 further includes accessing (at 204), by a first entity (e.g. 112) in the system, data in the data structure using a first interface (e.g. 108) designed for accessing data stored in the data structure in the memory of the system.

[0039] The process 200 further includes accessing (at 206), by a second entity (e.g. 118) in the system, data in the data structure using a persona object (e.g. 114) that presents to the second entity a second interface (e.g. 116) designed for accessing data stored external to the memory.

[0040] The persona object 114 may be an object created in the memory 102 when it is requested, and the persona object 114 may contain a reference to the associated data structure 106. In some examples (e.g. when the data structure 106 is a primitive data structure whose representation is smaller than a reference would be), the persona object 114 may contain a copy of the data structure 106 rather than a reference to the data structure 106. In some examples, the persona object 114 and its associated data structure 106 may be the same object, with the first (data structure) interface 108 and second (persona) interface (116) being two interfaces supported by the data structure 106.

[0041] The set of persona objects 114 that may be associated with a data structure 106 may be specified in various ways. In some examples, as described above, the data structure 106, via its implementation class, may support one or multiple second interfaces 116 (e.g. a data storage interface or a data structure interface) in addition to its primary data structure interface or interfaces 108. In such examples, a reference to a persona object 114 supporting one of these second interfaces 116 associated with the data structure 106 can be obtained simply by returning an appropriately typed reference to the data structure 106.

[0042] In some examples, the persona object 114 may be a distinct object (sometimes called a “proxy” or an “adaptor”) that holds a reference to the data

structure 106 and which implements (e.g. by means of machine readable instructions) the operations 117 declared by the second interface 116 in terms of one of the first interfaces 108 of the data structure 106. In some examples, the first interface 108 may include an operation 110 to create a proxy persona object 114 referring to the data structure 106 on which the operation 110 was invoked, where the persona object 114 supports a specific second interface 116. For example, a list data type may provide an operation for creating a persona object 114 that supports a file interface or, more specifically, a CSV file interface (e.g. a file interface in which each of the lines of the file is a string in CSV format) whenever the elements of the list are of a data type which provides the ability to convert the elements into strings or, more specifically, strings in CSV format. In other examples, the system 100 may provide a mechanism to associate the first interface 108 and the second interface 116 external to the first interface 108, allowing persona objects 114 supporting the second interface 116 to be created on demand and associated with any data structures 106 supporting the first interface 108. This association may be supplied as part of the system 100, as part of the specification (e.g. as part of the source code) of a program specifying the data structure 106, as part of a program linked with or otherwise associated with the second entity 118, by an operation invoked by a program, or by other mechanisms. In still further examples, the manner for creating a persona object 114 of a particular type referring to a particular data structure 106 may be associated (e.g. by any of the foregoing mechanisms) with the data structure 106 itself or with a collection that includes the data structure 106. A given data structure 106 may be associated with multiple different types of persona objects 114, and different types of persona objects 114 may be specified by different mechanisms. In some examples, a system may provide for rules by which the specification may be selected when the same type of persona object 114 is associated by different mechanisms. For example, the system may prioritize specifications associated with a data structure 106 itself over those associated with a data type 108 supported by the data structure 106.

[0043] The following provides various examples of operations (e.g. 117) that can be included in the second interface 116 presented by the persona object 114 of Fig. 1.

In some examples, the operation 117 can be a retrieval operation. If the second interface 116 is a file interface, then the retrieval operation may be an operation to read (e.g. a line) from a file. If the second interface 116 is a database interface, then the retrieval operation may be a select operation to identify data from one or more database tables. If the second interface 116 is a key-value interface, then the retrieval operation may be a lookup operation to lookup a value associated with a key in a key-value store.

[0044] Fig. 3 is a flow diagram of an example process 300 regarding use of a retrieval operation in the second interface 116 of the persona object 114 of Fig. 1. The process 300 includes receiving (at 302), from the second entity 118 via the persona object 114, a retrieval request. A request is received via a persona object if the request is produced in response to invocation of an operation on the persona object. The retrieval request is received via the persona object 114 in response to a call of the retrieval operation. The retrieval request is to retrieve data contained within the data 104 in the in-memory data structure 106. The process 300 associates (at 304) the retrieval request with a portion of the in-memory data structure 106, where the “portion” can refer to a data element or multiple data elements (e.g. further data structures 106) stored in, referred to by, or otherwise associated with the in-memory data structure 106. The association of the retrieval request with the portion of the in-memory data structure 106 can be an association of an address included in the retrieval request with data element(s) stored at memory locations identified by the address.

[0045] The process 300 returns (at 306), by the retrieval operation to the second entity 118, a value object based on the portion, where the value object can contain or be otherwise computed based on the data of the portion of the in-memory data structure 106 associated with the retrieval request. A “value object” can refer to a data structure 106, a primitive data object (e.g., a number, character, or Boolean value), a proxy object 114, or any other value that satisfies the criteria imposed by the retrieval operation 117. As an example of a computed value object, if the data structure 106 is a list (e.g. an array list) of records (stored by reference in the underlying array), each of which contains a name (as a string) and a score (as an integer), the interface 116 of the persona object 114 is that of a comma-separated values (CSV) file, and if the second entity 118 has already invoked the retrieval operation 117 twice, the third invocation of the retrieval operation 117 would cause process 300 to associate (at 304) the retrieval request with the third record referred to in the underlying array and the value object returned (at 306) may be a string including the record’s name, a comma, and the decimal representation of the record’s score.

[0046] When multiple entities in a system (e.g. programs, applications operating system processes, or threads) are allowed to concurrently access the same data structure 106, it is often desirable to shield entities from modifications to the data structure made by other entities in order to prevent the entities from being prematurely exposed to incomplete modifications or to be able to see a consistent and unchanging view of the data structure. Mechanisms to support this may be generically termed “isolation contexts” and refer to environments that processes may execute within. In general, modifications to a data structure made within an isolation context will not be visible to executions outside of the isolation context until the modifications are successfully published from the isolation context. Generally, when the modifications are successfully published, such publication happens atomically, with all modifications becoming visible outside the isolation context at the same time. If the isolation context is a snapshot isolation context, modifications to the data structure made outside of the isolation context will similarly not be visible to executions inside the isolation context, but rather all accesses made within the

isolation context will see the data structure as it was when the snapshot isolation context was created, except for modifications made within the isolation context.

[0047] Computations on a shared data structure may be isolated within corresponding isolation contexts. Due to the computational isolation, machine executable instructions, or program code, executing within the isolation contexts may concurrently make modifications to a data structure that is shared by the contexts. A "data structure" refers to an organization of one or multiple units of data, which are stored in one or multiple storage locations. In some examples, an isolation context may be used to access (e.g. read or modify) multiple data structures. Each isolation context may present to program code a corresponding "view" of the data structure, where the "view" refers to the value(s) that the isolation context reads for corresponding properties of the data structure. A single isolation context may be associated with multiple views. As examples, a "property" may be a location associated with the data structure (e.g. a field within a record or an index of an array), a structural property of the data structure (e.g. a number of elements in a list), or a relationship that is associated with the data structure (e.g. an association of a value with a key in a map).

[0048] Although the isolation contexts create computational isolation, there are mechanisms by which an isolation context may transfer information to another isolation context. One way a first isolation context may transfer information to a second isolation context is for the first isolation context to publish. "Publishing" an isolation context refers to combining or merging the view of the publishing isolation context with the view of another isolation context so that the views are the same at the time of publication. A given publication attempt, however, may not succeed due to one or multiple conflicts. In general, a "conflict" refers to a reason for the publication not to occur. An example of a conflict is the existence a modification made to a data structure (e.g. a change to a field of a record) within the isolation context that is the target of the publication attempt when a similar modification (e.g., a change to the same field of the same record) was made within the isolation context being published or when a similar value (e.g. the value of the same field of the same

record) was read within the isolation context being published and a modification to the same data structure or another data structure was made that may have been based on the value that was read.

[0049] In more specific examples, a system can include context management resources, which the system uses to create and manage isolation contexts, and manage publications by the isolation contexts. In general, an isolation context is a mechanism by which executing program code may isolate itself from other executing program code. As a more specific example, a program code may be an entire application or program or may be part of such an application or program, such as a thread. At any given point, the program code may be working in a prevailing (also referred to as the "current" or "working") isolation context. In a system that includes multiple threads, each thread may have its own prevailing isolation context, and different threads executing program code at the same time may be working in different prevailing isolation contexts. In some examples, when a parent thread creates a child thread, the prevailing isolation context in the child thread when it begins executing is that of the parent thread (e.g. at the time the child thread was created).

[0050] In some examples, properties of a data structure may be represented by multiple simultaneous value (MSV) objects. In general, an MSV object represents a property of a unit of data and may concurrently, or simultaneously, have multiple, alternative values. For example, a particular logical location, such as a field of a record or a slot in an array, may be represented by an MSV object, such that the field or slot has different, alternative values, when seen through different views.

[0051] A given isolation context may be associated with one or multiple views that may be used when accessing one or multiple MSV objects. Moreover, multiple isolation contexts may be associated with multiple views of a given MSV object. The values associated with different views in a given MSV object may be isolated from each other. In this manner, as part of this isolation, a request to determine a value for the MSV object in a given view may result in the return of one of the MSV object's

alternative values; and assigning the value of the MSV object in a given view may not affect the value of the MSV object in another view.

[0052] In alternative examples, instead of using isolation contexts as described above, a database-style transaction can be used instead of an isolation context. A transaction can refer to an operation or a collection of operations that is (are) performed in a system in response to a request. Modifications to data in a transaction can be committed or rolled back. If the transaction is successfully performed, then the transaction can be committed such that its results are written to persistent storage and made available to other entities of the system. If the transaction experiences a failure, then a rollback is performed to bring the data back to a state prior to the beginning of the transaction. In such examples, committing a transaction is a form of publishing an isolation context.

[0053] In some examples, the system 100 may ensure that retrieval operations are insensitive to concurrent modifications to the data structure 106 other than by the second entity 118 using the persona object 114.

[0054] Fig. 4 shows an example process 400 in which it is assumed that the second interface 116 includes a retrieval operation (to retrieve data, such as a file or other type of data) and an open operation (to open a container of data, such as a file or other type of data).

[0055] The process 400 includes receiving (at 402), from the second entity 118 via the persona object 114, an open request that is in response to invocation of the open operation in the second interface 116. In response to the open request, an isolation context is established and configured as a snapshot isolation context. Subsequent retrieval requests via the persona object 114 will be executed within this isolation context. After receiving the open request, the process 400 modifies (at 404) the in-memory data structure by another entity without reference to the persona object. This other entity can be the first entity 112 or a third entity, which is able to use an in-memory data structure interface (e.g. 108) without referencing (i.e. without using) the persona object 114. The modification at 404 happens outside the isolation context

and may be executed in a different execution context (e.g. thread or operating system process) from the other steps in process 400.

[0056] Since the modification operations performed with respect to the in-memory data structure 106 are being performed outside of a snapshot isolation context, the modifying of the in-memory data structure 106 by the other entity is not made visible to operations performed inside of the snapshot isolation context.

[0057] After the modifying of the in-memory data structure 106 has occurred, the process 400 receives (at 406), from the second entity 118, a retrieval request in response to invocation of the retrieval operation in the second interface 116.

[0058] The process 400 returns (at 408), to the second entity 118, a value object in response to the retrieval request, where the returned value object is unaffected by the modifying of the in-memory data structure 106 by the other entity. In other words, the returned value object may include an older version of data in the in-memory data structure 106 that has been modified by the other entity—the returned value object does not contain the newer version of the data as modified by the other entity. As a result, the second entity 118 can read data of the in-memory data structure 106 without having to worry about modifications to the in-memory data structure 106 by another entity while the second entity 118 is reading the data. The returned value object can contain (or be based on) the data of the portion of the in-memory data structure 106 associated with the retrieval request.

[0059] In alternative examples, the second entity 118 may specify (e.g. as a parameter supplied as part of an invocation of an open operation) that the second entity 118 wants to read the data in the in-memory data structure 106 in a live manner, where changes to the data in the in-memory data structure 106 are made visible to the second entity 118. In such an example, a snapshot isolation context may not be created.

[0060] Fig. 5 depicts an example process 500 that can be performed to perform modification of data in the in-memory data structure 106. It is assumed that the

second interface 116 includes a modification operation that can be invoked by the second entity 118 to modify data. The process 500 includes receiving (at 502), from the second entity 118 via the persona object 114, a modification request that is responsive to invocation of the modification operation in the second interface 116.

[0061] The process 500 determines (at 503) whether the modification request is a modify request or an insert request. A modify request can be a request to write (e.g. a line) to a file (where the write does not occur at the end of the file and therefore replaces existing data), for example. In other examples, the modify request can be a request to update a row or rows in a database table, or a request to put a new value with a key in a key-value store.

[0062] An insert request is a request to insert new data, such as appending a line to a file, adding a row to a database, or associating a value with a new key in a key-value store.

[0063] The determination (at 503) may be made, e.g. based on the operation invoked, based on data supplied (e.g. as arguments) along with the invocation of the operation, or based on the current state of the persona object 114 or the data structure 116. For example, an operation to associate a value with a key invoked on a persona object 114 associated with a key-value store interface may be a modify request if the data structure 106, when viewed as a key-value store already has a value associated with the given key and an insert request otherwise.

[0064] In response to determining (at 503) that the modification request is a modify request, the process 500 associates (at 504) the modify request with a portion of the in-memory data structure 106. The process 500 then modifies (at 506) the portion of the in-memory data structure 106, according to the modify request. For example, if the data structure 106 is a linked list of records each containing a name and a score, where the list is of length 10, the persona object 114 is presenting the interface of a CSV file, the persona object 114 maintains a position indication (e.g. a cursor) pointing to the third record in the list as the next element to be retrieved or modified, and the modification request is a request to write the string "Bob,20", the effect of

performing task 506 may be to modify the third record in the list such that its name becomes “Bob” and its score becomes 20.

[0065] In response to determining (at 503) that the modification request is an insert request, the process 500 creates (at 508), according to the insert request, a second data structure in the memory 102. The created second data structure can include new data to be inserted by the insert request. The process 500 then adds (at 510) the second data structure as a portion of the in-memory data structure 106.

Continuing the previous example, if the position indication indicates the end of the list, the same modification request to write the string “Bob,20” may cause (at 508) a new record to be created, with a name of “Bob” and a score of twenty, and this new record may (at 510) be appended to the end of the list.

[0066] Modification of the data in the in-memory data structure 106 can be performed by the second entity 118 in an isolation context, which is created in response to invocation of an open operation in the second interface 116. This ensures that any modifications to data in the in-memory data structure 106 made via the persona object 114 are not visible to other entities except via the persona object 114. The system renders (by creating an environment that is isolated from other entities except via the persona object 114) any alteration to the in-memory data structure 106 responsive to a modification request not visible to another entity except via the persona object 114. Note that any other entity that accesses the in-memory data structure 106 via the persona object 114 will be able to see alterations of the in-memory data structure 106 requested by the second entity 118.

[0067] Once the modifications of the in-memory data structure 106 requested by the second entity 118 are complete, the second entity 118 can invoke a close operation in the second interface 116, which causes a close request to be invoked subsequent to the alterations of the in-memory data structure 106. Invoking the close operation triggers an attempt to publish the isolation context which, if successful, renders the alterations of the in-memory data structure 106 made by the second entity 118 responsive to preceding modification requests visible to other entities, without such other entities having to use the persona object 114.

[0068] In some examples, to further improve efficiency for some legacy interfaces, a result of an operation of the second interface 116 can be returned to the second entity 118 as a response persona object. A response persona object includes information and operation(s) that allows for the result of an operation of the second interface 116 to be presented to the second entity 118. The second interface 116 associated with a response persona object 114 may be a data structure interface, but it can be one whose semantics are specialized for being the return value of an operation associated with a data storage interface.

[0069] As an example, consider the scenario of a legacy entity reading a CSV file, where each line of the file is assumed to contain a name (a string) and the decimal representation of a score (an integer), the two fields separated by a comma. An example approach would be, for each line of the file:

(1) read the line as a string,

(2) use a “split” operation to obtain an array of strings, with the first string in the array being the name and the second the decimal representation of the score, and

(3) parse the second string in the array to obtain the score as an integer.

[0070] If the CSV file is a proxy object 114 presenting a CSV file interface and associated with a list of records, each containing a name (as a string) and a score (as an integer), the task (1) above may involve an invocation of process 300 in Fig. 3, where the line read as the string may be the value object returned (at 306). If the portion identified at 304 is a record whose name is “Fred” and whose score is 12, rather than actually creating the string “Fred,12”, which would involve converting the integer score to a string and concatenating several strings, the value object can be a response persona object 114 associated with the record, whose interface 116 is a string data type but whose implementation of that interface had a specialized version of the “split” operation that directly created an array containing the record’s name and the decimal representation of the record’s value. If any other operations of the

string are invoked, as is not expected, the string (e.g. "Fred,12") can be created and cached, and these other operations can be implemented by delegating to the string. If further efficiency is desired, the result of the "split" operation itself can be a response persona object 114 associated with the record, but this response persona object 114 presents an array-of-strings interface, implemented such that an operation to retrieve the size will return the value 2, an operation to retrieve the first element of the array will return the record's name, and an operation to retrieve the second element of the array will return the decimal representation of the record's score. For further efficiency, this last value object may itself be a response proxy associated with the record or with the record's core and presenting a string interface, but with an implementation of the "parse as integer" operation that directly returns the record's score. With these response proxies, the expected loop described above can be executed and the name and score obtained without constructing any intermediate strings.

[0071] Fig. 6 illustrates an example process 600 that uses a response persona object to return a result of an access of the in-memory data structure 106. The process 600 receives (at 602), from the second entity 118 via the persona object 114, a request responsive to invocation of an operation in the second interface 116. The process 600 associates (at 604) a result of the request with a portion of the in-memory data structure 106 that is according to a first data type. The process 600 returns (at 606) the result of the request as a second persona object associated with the portion of the in-memory data structure 106, where the second persona object presents an interface for a second data type different from the first data type.

[0072] In accordance with further implementations, as shown in Fig. 7, a file system 700 (which can be associated with the system 100, for example) can be provided to allow legacy entities (e.g. second entity 118) to obtain persona objects 114 that present file interfaces 116. The file system 700 can associate hierarchical names (e.g. names within directories or folders 710 that are themselves identified by hierarchical names) with pseudo-files 702 that represent the ability to access specific in-memory data structures 706 (e.g. similar to the data structure 106 of Fig. 1) in the

memory 102 using specific data storage interfaces 116 (e.g. a file interface such as a CSV file interface). In some examples, the pseudo-files 702 may be objects in the memory 102. In other examples, the pseudo files 702 may be logical constructs identified upon attempted use. The file system 700 may further include a mapping 704 that specifies (e.g. by rule) an association between the pseudo-files 702 and respective in-memory data structures 706. In some examples, the file system 700 can be made available to a legacy entity by means of the `FileSystemProvider` class provided by the Java programming language, or some other class or technique provided by a different programming language. In other examples, the file system 700 can be the file system of an operating system. In some examples, the file system 700 may contain both pseudo-files 702 and regular files (e.g. files stored as sequences of bytes on a data storage devices). In other examples, files and pseudo-files 702 may be accessed via separate file systems 700.

[0073] The system 100 may contain a namespace that maps names to data structures 706, where a name can be in the form of a string (of characters), or more generally, in the form of objects. The presence of an association between a name and a data structure 706 in the namespace may imply an association between a name and a pseudo-file 702 in the file system 700, where the pseudo-file 702 may represent access to the data structure 706 via a data structure interface supported by some persona object 712 that can be associated with the data structure 706. For example, if the name “Customers” is associated by the namespace with an array of records and it is known how to construct a CSV file persona object 712 from an array of records, the file system 700 may contain a pseudo-file 702 associated (e.g. via the mapping 704) with the array and also associated (e.g. via the file system 700) with a name based on “Customers”. For example, the name of the pseudo-file might be “customers.csv” in some directory, indicating both the mapped data structure (e.g. “customers”) and the interface type (e.g. “csv”). If the same data structure 706 is associated by the namespace with multiple names, this may result in multiple pseudo-files 702 being associated in the file system 700 with the data structure 706.

[0074] In some examples, the system 100 may provide mechanisms for a program to explicitly (e.g. by calling a function) associate a particular data structure 706 with a particular pseudo-file 702 having a provided name in the file system 700. In some examples, the system 100 may provide mechanisms for a person to explicitly (e.g. by entering a command at a terminal window) associate a particular data structure 706 with a particular pseudo-file 702 having a provided name in the file system 700. In some examples, the system 100 may provide mechanisms for a program or person to specify a rule by which the file system may be able to infer the existence of a pseudo-file 702 given the presence of a data structure 706.

[0075] In some examples, a data structure 706 may be associated (e.g. by the mapping 704) with multiple pseudo-files 702, and some or all of the pseudo files 702 may be associated with different pseudo-file types, where a pseudo-file type represents a particular data storage interface 116 supported by an associated persona object 712 or a particular semantics for the way in which an associated persona object 712 implements its associated storage interface 116. For example, a data structure 706 that is a list of records may be associated with a pseudo-file of type CSV file, where this type indicates that an associated persona object 712 will behave as though the pseudo-file 702 contained a sequence of lines each containing comma-separated values. The same data structure may be associated with a pseudo-file 702 behaving as a file contain lines in tab-separated value (TSV) format, a pseudo-file 702 behaving as a file containing a JavaScript Object Notation (JSON) description, a pseudo-file 702 behaving as a file containing a Microsoft Excel spreadsheet, a pseudo-file 702 behaving as a file containing an Adobe Portable Document Format (PDF) rendering of the data contained in the data structure 706, or pseudo-files 702 of other types. The determination of which pseudo-file type is associated with a given pseudo-file 702 may be based on a name (e.g., a hierarchical name) associated with the pseudo-file 702 in the file system 700. In some examples, the determination may be made based on a portion of the name (e.g., pseudo-file names ending in ".csv" may be associated with CSV-type pseudo-files 702 while pseudo-file names ending in ".json" may be associated with JSON-type pseudo-files 702). In some examples, the determination may be based on a

directory 710 that is part of the hierarchical name of the pseudo-file 702 (e.g., all pseudo-files inside a directory named “reports” may be PDF-type pseudo-files).

[0076] A pseudo-file 702 may represent a directory (e.g. directory 710) and may itself contain pseudo-files or files. In some examples, if the data structure 706 associated with the pseudo file 702 is a map data structure, representing an association between keys objects of a key data type and value object of a value data type, the directory 710 associated with the pseudo-file 702 may contain further pseudo-files 702 associated with some or all of the value objects contained in the map data structure, with the names of the pseudo-files 702 in the directory 710 based on the associated key objects. In some examples, a directory 710 presented by a pseudo-file 702 associated with a data structure 706 may contain further pseudo-files 702 associated with different versions of the data structure 706 (e.g. snapshots of the data structure taken at different times) and the names associated with these further pseudo-files 702 may be based on information (e.g. a representation of a timestamp) used to identify a particular version of the data structure 706.

[0077] The following describes how a legacy entity, such as the second entity 118 of Fig. 1, can access data of an in-memory data structure 706 using a pseudo-file 702. A first in-memory data structure 706 is associated with a first pseudo-file 702 in the file system 700. As described above, this association may be implicit or explicit. The second entity 118 identifies the first pseudo-file 702 by interacting with the file system 700, e.g. by presenting to the file system 700 a name associated by the file system 700 with the first pseudo-file 702. In response to an attempt by the second entity 118 to access the first pseudo-file 702 (e.g. by invoking an “open” request referring to the first pseudo-file 702 in the file system 700), the second entity 118 receives a first persona object 712. The first persona object 712 may preexist the attempt to access the first pseudo-file 702 or it may be created in response to the attempt to access the first pseudo-file 702. The second entity receiving the first persona object 712 can refer to the second entity receiving a reference to the first persona object 712.

[0078] In further examples, the mapping 704 may specify that a given in-memory data structure 706 is associated with multiple pseudo-files 702 in the file system 700. In such examples, based on a first pseudo-file 702 being accessed by the second entity 118, a persona object can be selected from the multiple persona objects 712 to use for accessing the data in the given in-memory data structure 706.

[0079] The mapping 704 can be bidirectional, to be used for reading data from an in-memory data structure 706, and also to be used for writing data to an in-memory data structure 706. For example, an entity can write to a pseudo-file 702, and using the mapping 704, the file system 700 can determine that an effect of the write operation should be a modification to an in-memory data structure 706.

[0080] Fig. 8 is a block diagram of an example non-transitory machine-readable or computer-readable storage medium 800 that stores machine-readable instructions executable to perform various tasks as discussed herein. The machine-readable instructions include first receiving instructions 802 to receive invocation by a first entity of an operation in a first interface designed for accessing data stored in an in-memory data structure stored in a memory; first result providing instructions 804 to, responsive to the invocation of the operation, provide, via the first interface, a result of the operation to the first entity; second receiving instructions 806 to receive invocation by a second entity of an operation in a second interface within a persona object, the second interface presented to the second entity and designed for accessing data stored external to the memory of the system; and second result providing instructions 808 to, responsive to the invocation of the operation in the second interface, provide, via the second interface, a result of the operation in the second interface to the second entity.

[0081] The machine-readable instructions are executable on a processor (or multiple processors). A processor can include a microprocessor, a microcontroller, a core of a multi-core microprocessor, a co-processor, a graphics processing unit (GPU), a general-purpose GPU (GPGPU), a programmable integrated circuit, a programmable gate array, or another hardware processing circuit. The machine-readable instructions are also executable on a first processor if the machine-readable

instructions are written according to a specification for a second processor and the first processor is capable of emulating the second processor, where the emulation may be in hardware or software.

[0082] The storage medium 800 can include one or multiple different forms of memory including semiconductor memory devices such as dynamic or static random access memories (DRAMs or SRAMs), erasable and programmable read-only memories (EPROMs), electrically erasable and programmable read-only memories (EEPROMs) and non-volatile memories such as flash memories, memristor memories, resistive memories and spin-transfer torque memories; magnetic disks such as fixed, floppy and removable disks; other magnetic media including tape; optical media such as compact discs (CDs) or digital versatile discs (DVDs); or other types of storage devices. Note that the instructions discussed above can be provided on one computer-readable or machine-readable storage medium, or alternatively, can be provided on multiple computer-readable or machine-readable storage media distributed in a large system having possibly plural nodes. Such computer-readable or machine-readable storage medium or media is (are) considered to be part of an article (or article of manufacture). An article or article of manufacture can refer to any manufactured single component or multiple components. The storage medium or media can be located either in the machine running the machine-readable instructions, or located at a remote site from which machine-readable instructions can be downloaded over a network for execution.

[0083] In the foregoing description, numerous details are set forth to provide an understanding of the subject disclosed herein. However, implementations may be practiced without some of these details. Other implementations may include modifications and variations from the details discussed above. It is intended that the appended claims cover such modifications and variations.

What is claimed is:

- 1 1. A method comprising:
2 storing data in a data structure in a memory of a system comprising a
3 processor;
4 accessing, by a first entity in the system, data in the data structure using a
5 first interface designed for accessing data stored in the data structure in the memory
6 of the system; and
7 accessing, by a second entity in the system, data in the data structure using a
8 persona object that presents to the second entity a second interface designed for
9 accessing data stored external to the memory of the system.
- 1 2. The method of claim 1, wherein the second interface comprises an operation,
2 the method further comprising:
3 receiving, from the second entity via the persona object, a request to invoke
4 the operation;
5 associating a result of the request with a portion of the data structure having a
6 first data type; and
7 returning as the result of the request a second persona object associated with
8 the portion of the data structure, the second persona object presenting an interface
9 for a second data type different from the first data type.
- 1 3. The method of claim 1, wherein the second interface is one of a file interface,
2 a directory interface, a file system interface, a database interface, a database table
3 interface, and a key-value store interface.

- 1 4. The method of claim 1, wherein the second interface is a file interface, the
2 method further comprising:
3 associating the data structure with a first pseudo-file in a file system
4 associated with the system; and
5 receiving, by the second entity, the persona object in response to an attempt
6 to access the first pseudo-file.
- 1 5. The method of claim 4, further comprising:
2 associating the data structure with a plurality of pseudo-files in the file system;
3 and
4 selecting, based on the first pseudo-file, the persona object from a plurality of
5 persona objects associated with the data structure.
- 1 6. The method of claim 1, wherein the second interface comprises a retrieval
2 operation, the method further comprising:
3 receiving, from the second entity via the persona object, a retrieval request in
4 response to invocation of the retrieval operation;
5 associating the retrieval request with a portion of the data structure; and
6 returning, to the second entity, a value object based on the portion.
- 1 7. The method of claim 6, wherein the second interface further comprises an
2 open operation, the method further comprising:
3 receiving, from the second entity via the persona object, an open request in
4 response to invocation of the open operation; and
5 subsequent to receiving the open request and prior to receiving the retrieval
6 request, modifying the data structure by a third entity in the system without reference
7 to the persona object,
8 wherein the returned value object is unaffected by the modification.

- 1 8. The method of claim 1, wherein the second interface comprises a modification
2 operation, the method further comprising:
3 receiving, from the second entity via the persona object, a modification
4 request responsive to invocation of the modification operation; and
5 altering the data structure based on the modification request.
- 1 9. The method of claim 8, wherein the modification request is a modify request,
2 and wherein altering the data structure comprises:
3 associating the modify request with a portion of the data structure; and
4 modifying the portion according to the modify request.
- 1 10. The method of claim 8, wherein the modification request is an insert request,
2 and wherein altering the data structure comprises:
3 creating, according to the insert request, a second data structure in the
4 memory; and
5 adding the second data structure as a portion of the data structure.
- 1 11. The method of claim 8, wherein the second interface further comprises an
2 open operation and a close operation, the method further comprising:
3 receiving, from the second entity via the persona object prior to receiving the
4 modification request, an open request to invoke the open operation;
5 rendering an alteration to the data structure responsive to the modification
6 request not visible to entities in the system other than via the persona object;
7 receiving, from the second entity via the persona object subsequent to the
8 modification request, a close request to invoke the close operation; and
9 rendering an alteration to the data structure responsive to preceding
10 modification requests via the persona object visible to other entities in the system.

1 12. A system comprising:
2 a memory to store data in a data structure;
3 a first interface comprising a first operation designed to access data in the
4 data structure in the memory;
5 a persona object to present a second interface comprising a second operation
6 designed to access data stored external to the memory;
7 a first entity to access data in the data structure by invoking the first operation
8 in the first interface; and
9 a second entity to access data in the data structure by using the persona
10 object.

1 13. The system of claim 12, wherein the persona object is to return a result of the
2 access by the second entity in a response persona object that presents an interface
3 for a data type different from a data type of the data structure.

1 14. A non-transitory storage medium storing instructions that upon execution
2 cause a system to:
3 receive, invocation by a first entity of an operation in a first interface designed
4 for accessing data stored in an in-memory data structure stored in a memory;
5 responsive to the invocation of the operation, provide, via the first interface, a
6 result of the operation to the first entity;
7 receive, invocation by a second entity of an operation in a second interface
8 within a persona object, the second interface presented to the second entity and
9 designed for accessing data stored external to the memory of the system; and
10 responsive to the invocation of the operation in the second interface, provide,
11 via the second interface, a result of the operation in the second interface to the
12 second entity.

- 1 15. The non-transitory storage medium of claim 14, wherein the instructions upon
- 2 execution cause the system to receive, from a third entity, an indication of an
- 3 association of the persona object that is of a persona type with the in-memory data
- 4 structure.

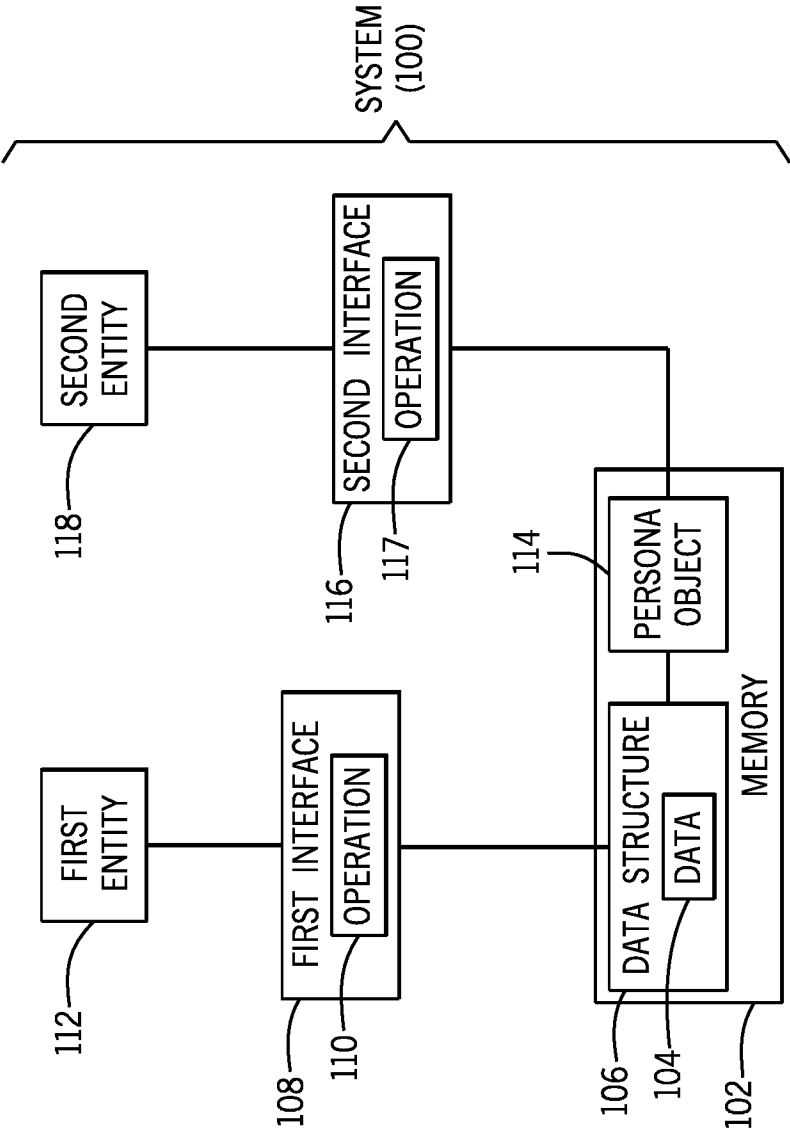


FIG. 1

2 / 8

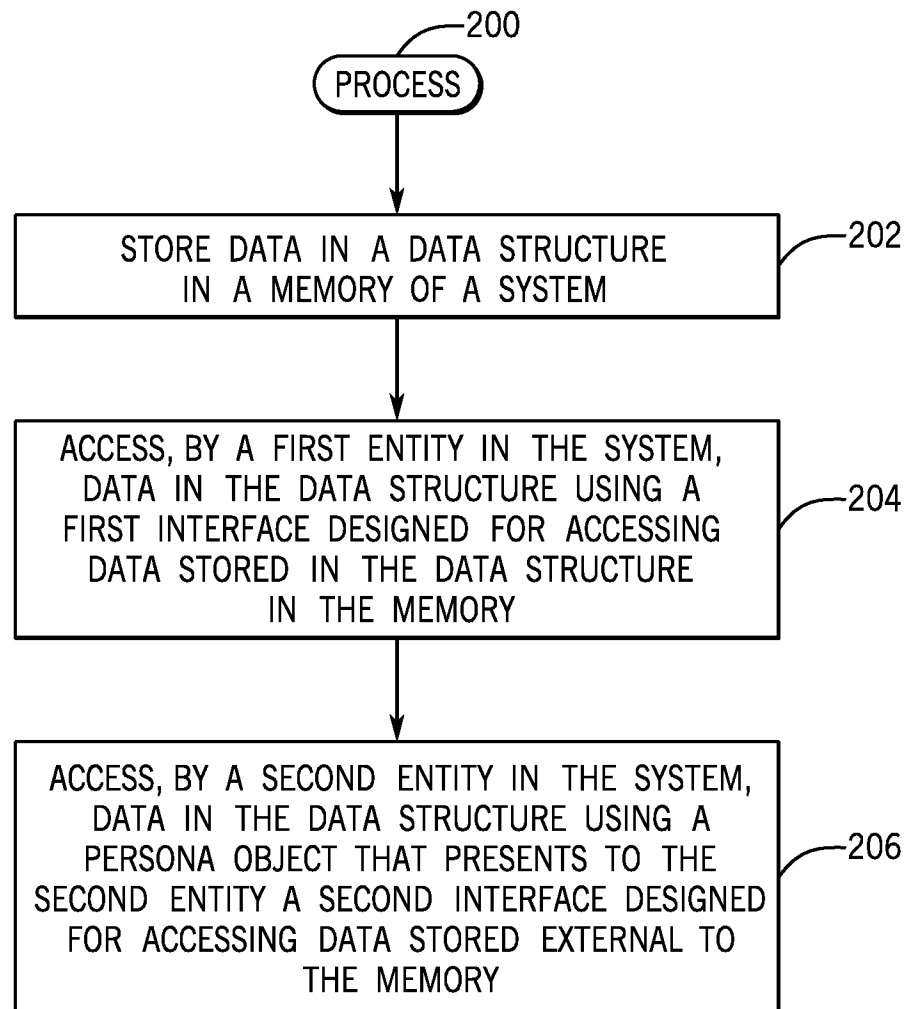


FIG. 2

3 / 8

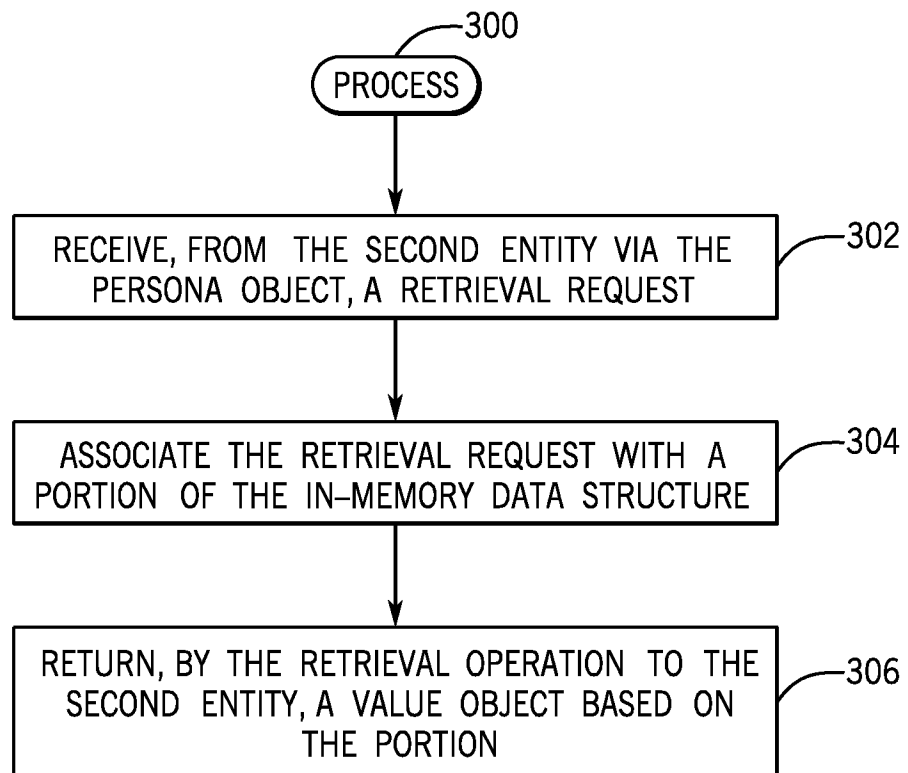


FIG. 3

4 / 8

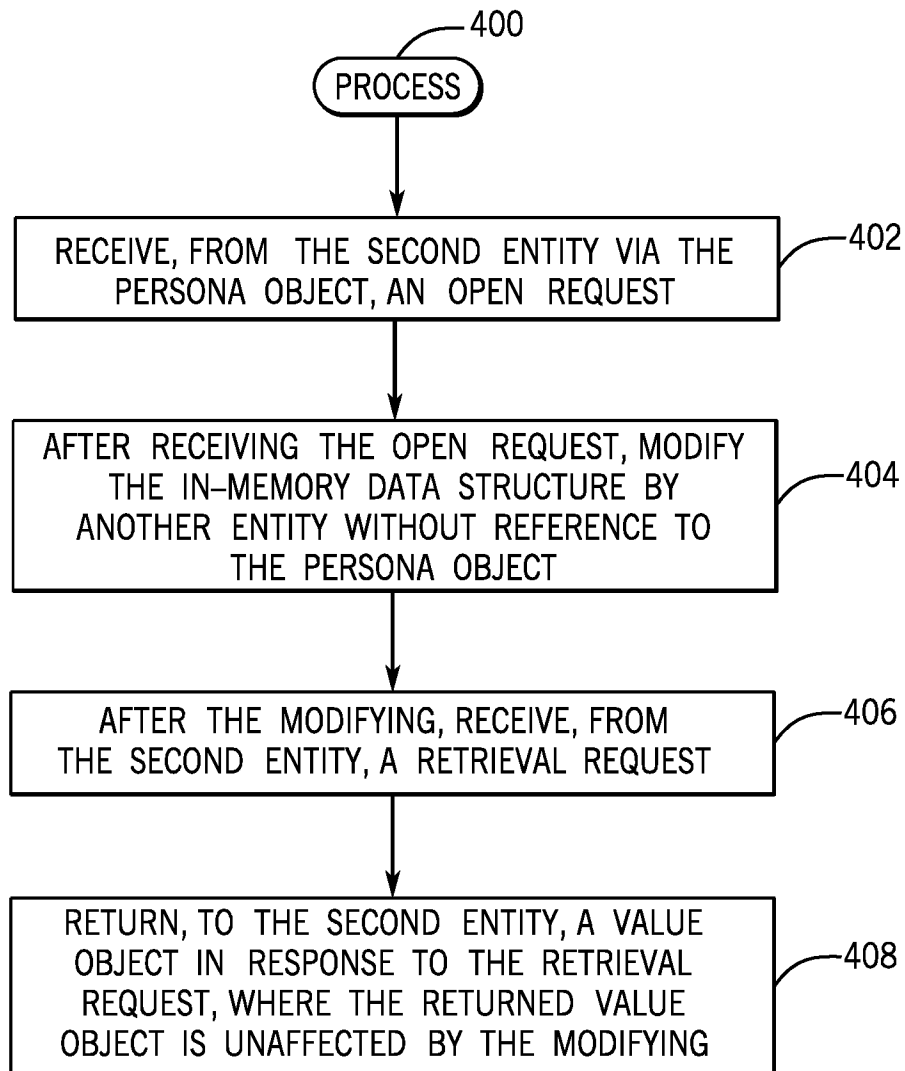


FIG. 4

5 / 8

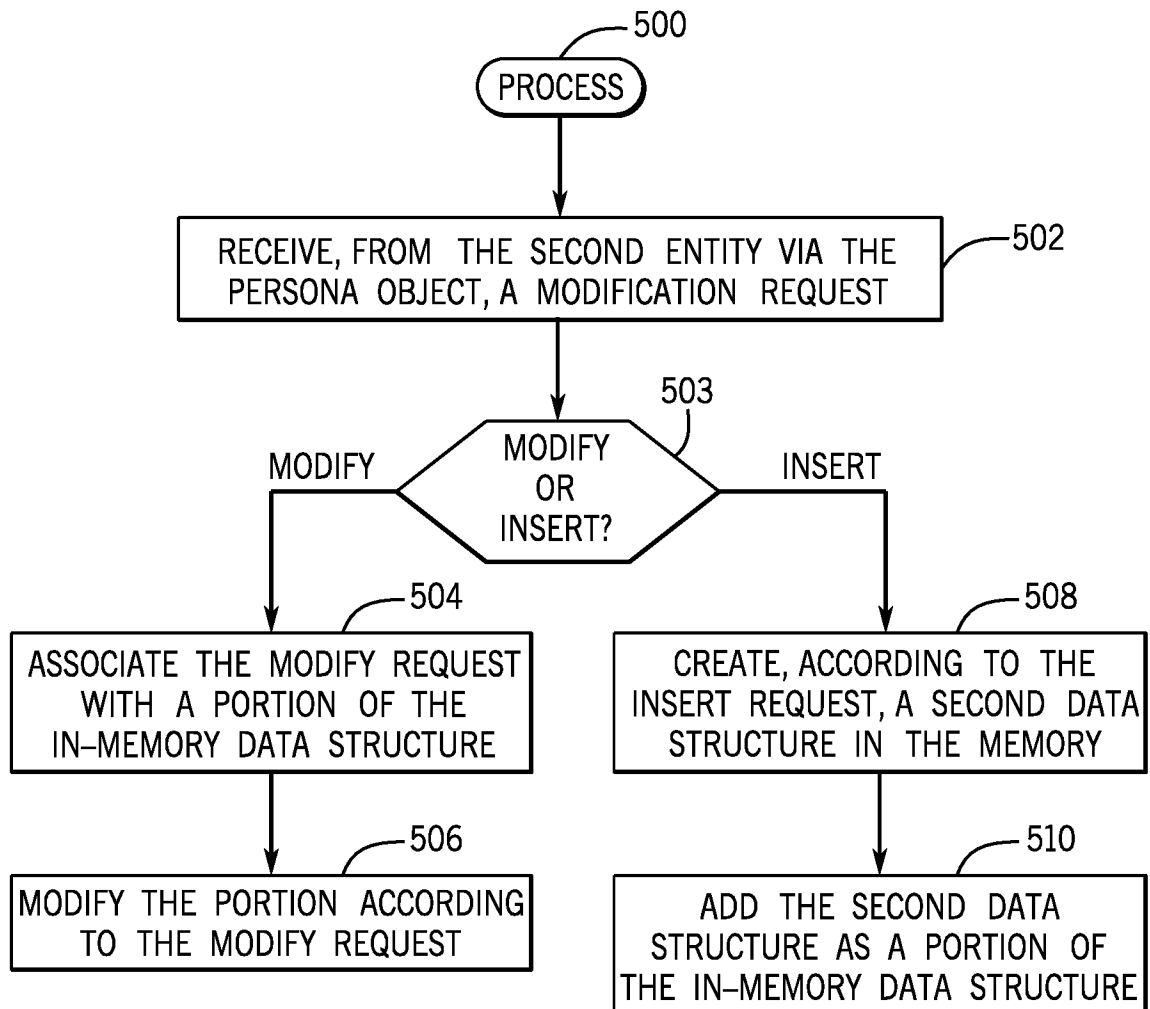


FIG. 5

6 / 8

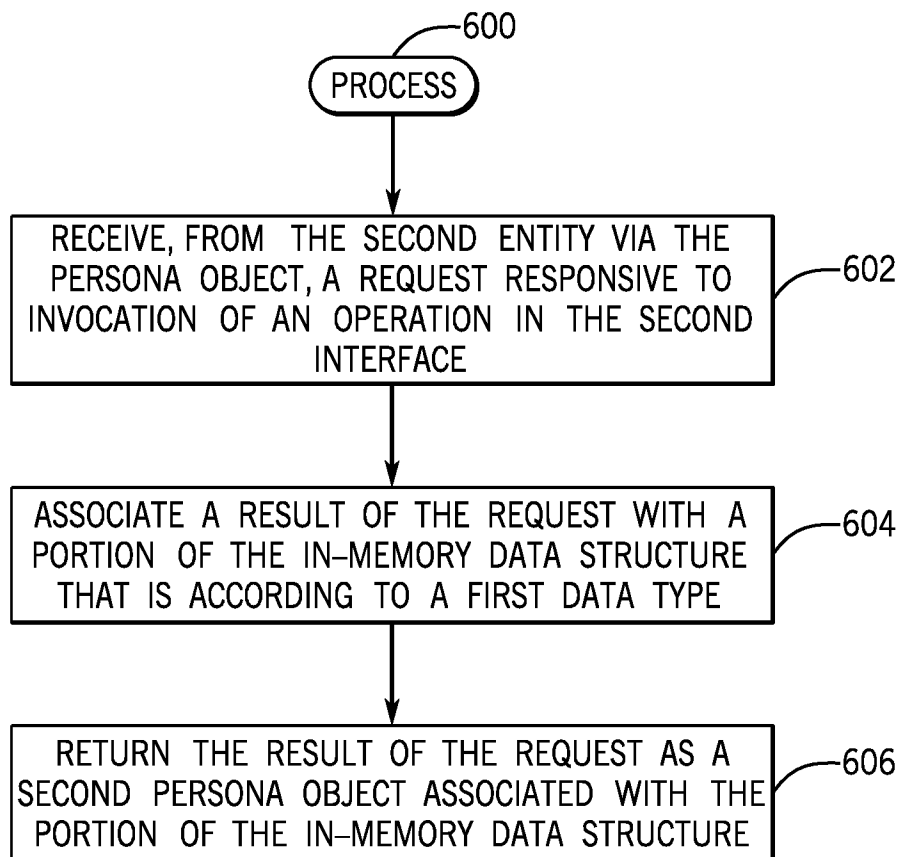


FIG. 6

7 / 8

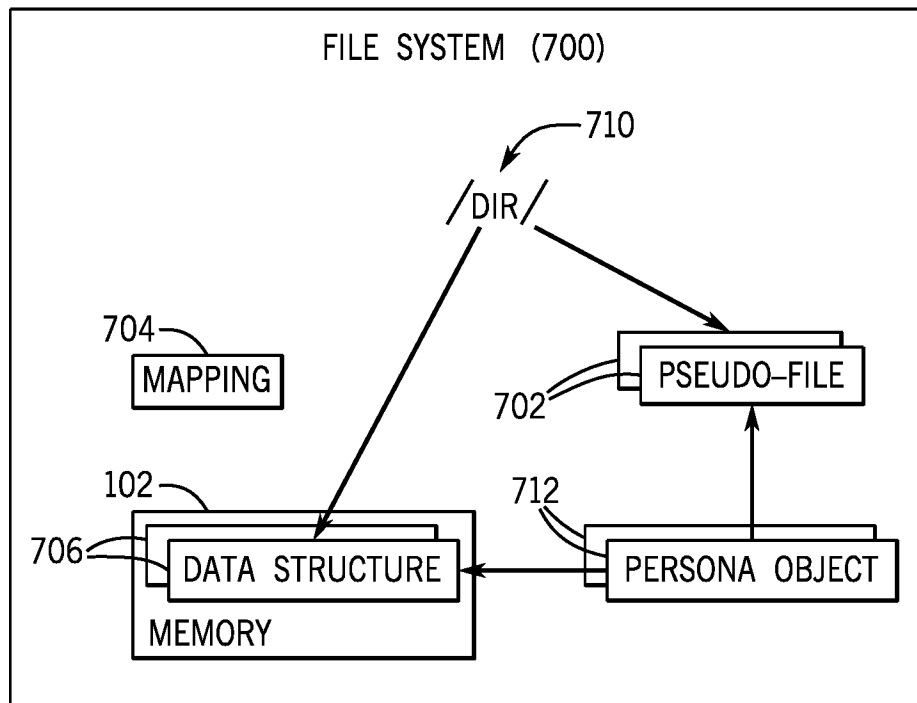


FIG. 7

8 / 8

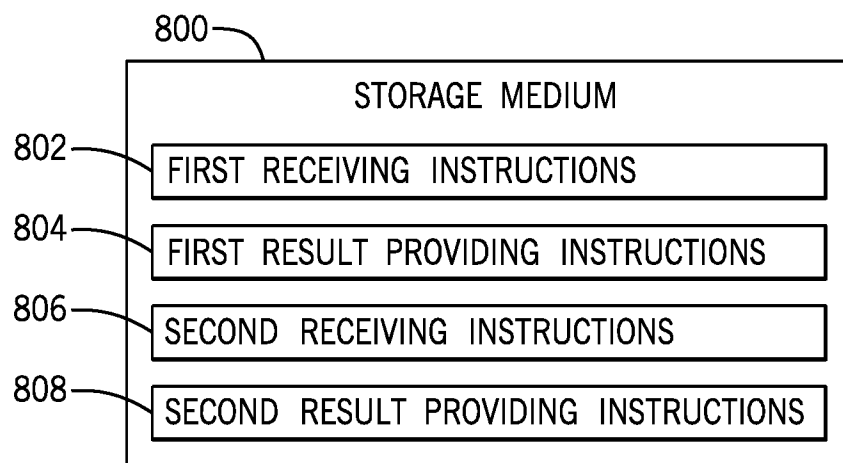


FIG. 8

A. CLASSIFICATION OF SUBJECT MATTER**G06F 17/30(2006.01)i**

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 17/30; G06F 3/00; G06F 7/00; G06F 17/60; G06F 9/455

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

Korean utility models and applications for utility models

Japanese utility models and applications for utility models

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

eKOMPASS(KIPO internal) & Keywords: data structure, in-memory, different, type, interface, persona, access, and similar terms.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2013-0066925 A1 (MICROSOFT CORPORATION) 14 March 2013 See paragraphs [0008], [0024]-[0025], [0030], [0033]-[0036], [0039], [0044], [0048], [0051]-[0054], and [0057]-[0058]; and figures 1A-1B and 3.	1-3, 6, 12-15
A		4-5, 7-11
A	US 7,136,859 B2 (MARK LUCOVSKY et al.) 14 November 2006 See column 1, lines 29-32; column 5, line 13 - column 6, line 26; claim 1; and figures 3-4.	1-15
A	US 2012-0159479 A1 (ALVIN CHARDON et al.) 21 June 2012 See paragraphs [0022]-[0025] and figure 1.	1-15
A	US 2002-0080192 A1 (NEAL J. KING et al.) 27 June 2002 See paragraphs [0056]-[0061] and figures 4-5.	1-15
A	WO 01-75724 A1 (PERSONA, INC.) 11 October 2001 See pages 8 and 10-14; and figures 2A-2B.	1-15



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

13 January 2017 (13.01.2017)

Date of mailing of the international search report

16 January 2017 (16.01.2017)

Name and mailing address of the ISA/KR

International Application Division

Korean Intellectual Property Office

189 Cheongsa-ro, Seo-gu, Daejeon, 35208, Republic of Korea

Facsimile No. +82-42-481-8578

Authorized officer

NHO, Ji Myong

Telephone No. +82-42-481-8528



INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2016/029236

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2013-0066925 A1	14/03/2013	CN 101952800 A	19/01/2011
		CN 101952800 B	14/01/2015
		EP 2260377 A2	15/12/2010
		EP 2260377 A4	12/12/2012
		JP 2011-515734 A	19/05/2011
		JP 5400068 B2	29/01/2014
		US 2009-0216778 A1	27/08/2009
		US 8307016 B2	06/11/2012
		US 8756257 B2	17/06/2014
		WO 2009-108426 A2	03/09/2009
		WO 2009-108426 A3	29/10/2009
US 7136859 B2	14/11/2006	AU 2002-244222 A8	24/09/2002
		AU 2002-258436 A8	15/10/2002
		EP 1370931 A1	17/12/2003
		EP 1370951 A2	17/12/2003
		EP 1370963 A2	17/12/2003
		EP 1370965 A1	17/12/2003
		EP 1370987 A1	17/12/2003
		EP 1370987 A4	04/10/2006
		EP 1370991 A1	17/12/2003
		EP 1379971 A2	14/01/2004
		EP 1379971 B1	01/02/2012
		US 2002-0133535 A1	19/09/2002
		US 2002-0174125 A1	21/11/2002
		US 2002-0184214 A1	05/12/2002
		US 2002-0184521 A1	05/12/2002
		US 2003-0023623 A1	30/01/2003
		US 2003-0061365 A1	27/03/2003
		US 2003-0097485 A1	22/05/2003
		US 2003-0101190 A1	29/05/2003
		US 2003-0115228 A1	19/06/2003
		US 2003-0131069 A1	10/07/2003
		US 2003-0131073 A1	10/07/2003
		US 2003-0131142 A1	10/07/2003
		US 2004-0024875 A1	05/02/2004
		US 2004-0199861 A1	07/10/2004
		US 2004-0199869 A1	07/10/2004
		US 2005-0273692 A1	08/12/2005
		US 2005-0278344 A1	15/12/2005
		US 2005-0278366 A1	15/12/2005
		US 2006-0036642 A1	16/02/2006
		US 2006-0161554 A1	20/07/2006
		US 2007-0083561 A1	12/04/2007
		US 6980993 B2	27/12/2005
		US 6985958 B2	10/01/2006
		US 7206788 B2	17/04/2007
		US 7284271 B2	16/10/2007
		US 7302634 B2	27/11/2007

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2016/029236

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
		US 7539747 B2	26/05/2009
		US 7548932 B2	16/06/2009
		US 7613702 B2	03/11/2009
		US 7613721 B2	03/11/2009
		US 7613722 B2	03/11/2009
		US 7664724 B2	16/02/2010
		WO 02-073339 A2	19/09/2002
		WO 02-073339 A3	19/09/2002
		WO 02-073392 A1	19/09/2002
		WO 02-073442 A1	19/09/2002
		WO 02-073454 A2	19/09/2002
		WO 02-073454 A3	21/11/2002
		WO 02-073466 A1	19/09/2002
		WO 02-073472 A1	19/09/2002
		WO 02-079921 A2	10/10/2002
		WO 02-079921 A3	10/10/2002
US 2012-0159479 A1	21/06/2012	CN 102591700 A	18/07/2012
		EP 2656210 A2	30/10/2013
		EP 2656210 A4	03/09/2014
		JP 2014-501408 A	20/01/2014
		JP 2016-197460 A	24/11/2016
		JP 5989001 B2	07/09/2016
		TW 201229805 A	16/07/2012
		US 2016-134721 A1	12/05/2016
		US 9116728 B2	25/08/2015
		WO 2012-087990 A2	28/06/2012
		WO 2012-087990 A3	18/10/2012
US 2002-0080192 A1	27/06/2002	US 6452614 B1	17/09/2002
WO 01-75724 A1	11/10/2001	AU 2001-51076 A1	15/10/2001