(51) International Patent Classification[7]:   G06F 12/00, 17/30, 17/50

(21) International Application Number: PCT/US00/12049

(22) International Filing Date: 2 May 2000 (02.05.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/137,355   3 June 1999 (03.06.1999)   US
09/557,329   24 April 2000 (24.04.2000)   US

(71) Applicant: SYCAMORE NETWORKS, INC. [US/US]; 10 Elizabeth Drive, Chelmsford, MA 01824-4111 (US).

(72) Inventor: CHAN, Dickson, K., T.; 2 Jack Rabbit Lane, Westford, MA 01886 (US).

(74) Agents: SCHURGIN, Stanley, M. et al.; Weingarten, Schurgin, Gagnebin & Hayes LLP, Ten Post Office Square, Boston, MA 02109 (US).

(81) Designated States (national): AU, CA, JP.

(84) Designated States (regional): European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published:
— With international search report.

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: SOFTWARE TRANSLATION USING METADATA
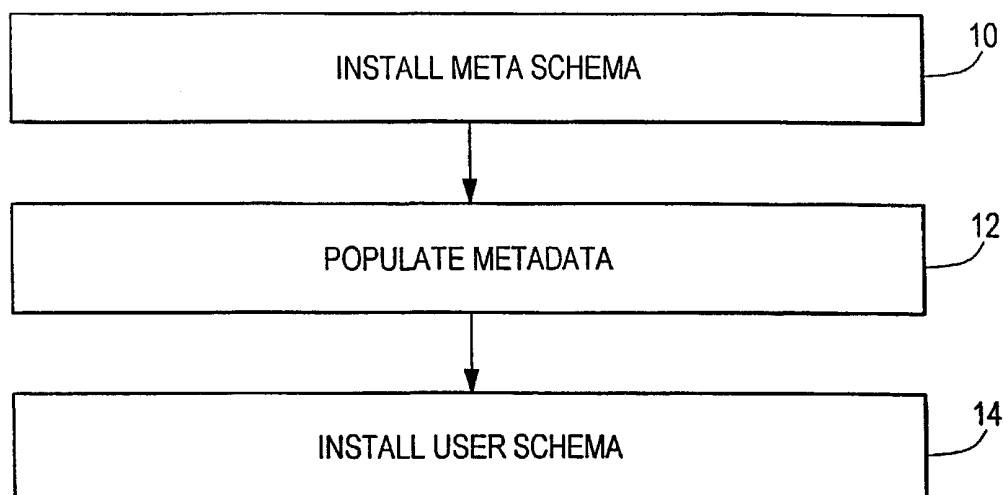
(57) Abstract: A system for supporting revisions to a software program, such as a network management program. All versions of software are forward and backward compatible. Version information is captured as part of the metadata used by the system. The meta schema entities for describing and storing metadata are installed [10], populated [12] with metadata reflecting for example a type of user application such as a network management system. User-data objects are then generated [14] for storing user data reflecting specific associated devices within a particular run-time environment. Each of the metadata entities and user-data entities are "version tagged" such that entries within these entities are indexed inpart based on a version tag identifying a specific software revision.

TITLE OF THE INVENTION

SOFTWARE TRANSLATION USING METADATA

5   CROSS REFERENCE TO RELATED APPLICATIONS

This application claims priority under 35 USC §119(e) to

provisional application serial number 60/137,355 filed

June 3, 1999.

10   STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR

DEVELOPMENT

N/A

BACKGROUND OF THE INVENTION

15

The present invention relates generally to computer

software development and more specifically to a system

and method for supporting software revision changes.

One of the persistent problems facing network

20   architects is managing the introduction of new features

and technology into the network. Traditional software

development processes result in systems that become more

difficult to change as new features are added or

existing features are modified. Due to the complexity

25   of synchronizing software revisions across the network

and the risks associated with introducing a new software

revision into the network, operators are generally

reluctant to attempt more than one major upgrade every

one to two years. While this ensures maximum stability

30   for the network, it can be a barrier to the introduction

of new functionality and technology.

Metadata has been used in existing systems to

describe the meaning and context of a piece or stream of

data.   Metadata has also been used to provide a structure and means for introducing new data.   For example, the number 100 may be considered a piece of data.   Alone, this data has no context.   However, a system may add metadata, such as the following text, to describe this piece of data: "Payroll data; a monetary amount in United States Dollars, expressed terms of dollars and cents, paid in bi-weekly increments."   In combination with such metadata, the number 100 has much more meaning.   Metadata has allowed existing systems to treat information generically, enabling software to be developed that focuses on the data processing problem without concern for the actual meaning of the data. Additionally, the meaning of a particular piece of data may be changed through adjustment of the appropriate metadata.   In the preceding example, the meaning of the data 100 could conveniently be changed to "Payroll data; a monetary amount in USD, expressed terms of dollars and cents, paid in weekly increments" by adjusting the associated metadata.   Neither the data element itself, nor the processes used to manipulate it must be changed. In this way, metadata provides tremendous flexibility to manipulate data in a non-disruptive way.   Using metadata allows software to be written in a generic fashion, enabling the software developer to concentrate on the overall design goals of the system (such as distributed processing) without being overly concerned with the actual data.

As described above, existing systems have used metadata as a higher form of description.   However, a significant drawback of existing systems arises in situations   where   certain   objects   and/or   object

-3-

attributes may change over time, in both nature and/or
value. Such circumstances may arise, for example, when
a network management system must be modified to reflect
an upgrade of a managed resource within the network

5   being managed, such as when a network application server
is upgraded to a new revision. In this type of
situation, one or more managed objects associated with
the resource being upgraded may change in terms of, for
example, the range of values which may be accepted for

10   an attribute, the methods which can be performed on an
attribute or object, and/or the fundamental
relationships between attributes or the managed objects
themselves. The effort required to accomplish system
modifications of this sort may require large numbers of

15   programmers to spend significant amounts of time
revising network management code to reflect the upgrade.

Accordingly, it would be desirable to have a system
which makes the introduction of new technology and
features to an existing software system more convenient

20   and less labor intensive. The system should further be
advantageously applicable to a network management
system.

BRIEF SUMMARY OF THE INVENTION

25

Consistent with principles of the invention, a
method and system for supporting revisions to a software
program are disclosed. In an illustrative embodiment,
the software program in which the present invention is

30   embodied is a network management program. The disclosed
system provides for an architecture or design affecting
a number of relationship data structures within the

software program.   Such relationship data structures describe relationships between types of data objects within the software program. For example, each element of each such relationship data structure may describe a

5   respective relationship between data objects (or "instances") of a first object type and data objects (or "instances") of a second object type. The architecture or design provided by the disclosed system may further affect a number of attribute data structures describing

10   attributes for data objects of various data object types.   Each element of each such attribute data structure, may, for example, describe an attribute of an associated object type.   The disclosed system establishes an association between a number of software

15   revision numbers, each associated with a respective revision of the software program, and each element in the relationships data structures, and potentially also with each element in the attributes data structures. In this way, the disclosed system provides a metadata,

20   which further describes the data, the relationships between the data, and the actions performed with the data, in terms of a revision of the software program using the data.   In an embodiment for a network management program, this technique advantageously

25   permits associated applications to be easily maintained and modified to meet changing requirements.

The disclosed system is described in connection with an illustrative embodiment, which provides a version independent network management system that

30   simplifies and facilitates network upgrades.   As disclosed herein, using the disclosed system, all versions of software are forward and backward

-5-

compatible. To achieve such version independence, version information is captured as part of the metadata used by the disclosed system. Additionally, as the metadata evolves over time, older elements are not deleted, but instead are updated by the disclosed system to indicate their relationship to any newer components.

Further, in a client-server embodiment of the disclosed system, a server determines a version of data that the client understands, as well as a matching version of metadata. The server then communicates to the client using an information model defined by the matching version of the metadata. In this way, the disclosed system enables "hitless" upgrades and eliminates the requirement to synchronize software across a network when introducing a new revision. Additionally, managed elements can be upgraded systematically over time. During the upgrade period, such elements continue to communicate with all devices at the current revision level until such time as the new revision level is introduced to the relevant node.

The forward and backward compatibility provided by the disclosed system make the introduction of new versions of software relatively risk-free and painless. Accordingly, network upgrades can be done anytime, eliminating any requirement for midnight upgrades or taking the network down for any period of time. In the event that a new version of software does not perform as desired, the network operator can easily return to the previous version, for example with a simple "point and click" operation through a graphical user interface (GUI) or the equivalent.

-6-

BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWING

The invention will be more fully understood by reference to the following detailed description of the invention in conjunction with the drawings, of which:

5   Fig. 1 is a flow chart illustrating steps performed in connection with operation of the disclosed system;

Fig. 2 is a block diagram showing a number of meta-schema entities for storing meta-data;

Fig. 3 shows simultaneous inter-operation between a
10   number of software entities associated with a first revision of a software program and a number of software entities associated with a second revision of the software program, as provided by the disclosed system;

Fig. 4 shows a table of column definitions for a
15   meta-schema entity used to store versions of meta-data associated with a number of software program revisions;

Fig. 5 shows a table of column definitions for a meta-schema entity used to store a number of meta-data object types;

20   Fig. 6 shows a table of column definitions for a meta-schema entity used to describe a number of meta-data attributes;

Fig. 7 shows a table of column definitions for a meta-schema entity used to store a number of meta-data
25   enumerated attribute values;

Fig. 8 shows a table of column definitions for a meta-schema entity used to store a number of meta-data conditions.

Fig. 9 shows a table of column definitions for a
30   meta-schema entity used to store a number of meta-data "identity" relational attributes;

-7-

Fig. 10 shows a table of column definitions for a meta-schema entity used to store a number of meta-data "contain" relationships;

Fig. 11 shows a table of column definitions for a meta-schema entity used to store a number of meta-data "pool" relational attributes;

Fig. 12 shows a table of column definitions for a meta-schema entity used to store a number of meta-data "consume" relationships;

Fig. 13 shows a table of column definition for a meta-schema entity used to store a number of meta-data "pointer" relational attributes;

Fig. 14 shows a table of column definitions for a meta-schema entity used to store a number of meta-data "connection" relationships;

Fig. 15 shows a table of column definitions for a user data entity used to describe a number of users;

Fig. 16 shows an illustrative configuration of meta-data entries within the meta-schema entity used to describe a number of meta-data objects;

Fig. 17 shows a table of default column definitions to be used within user data entities;

Fig. 18 shows an illustrative configuration of meta-data entries within the meta-schema entity for storing a number of meta-data attributes;

Fig. 19 shows tables of column definitions for user data entities describing user devices of associated device types;

Fig. 20 shows an illustrative configuration of meta-data entries loaded into the meta-schema entity used to store "identity" relational attributes;

Fig. 21 shows tables of column definitions for user data entities describing user devices of associated device types;

Fig. 22 shows an illustrative configuration of meta-data entries within the meta-schema entity for describing a number of meta data "pool" relational attributes;

Fig. 23 shows tables of column definitions for user entities describing associated user devices;

Fig. 24 shows a table of column definitions for a user data entity used to describe "contain" relationships between user devices;

Fig. 25 shows a table of column definitions for a user data entity used to describe "consume" relationships between user devices; and

Fig. 26 shows a table of column definitions for a user data entity used to describe "connect" relationships between user devices.

DETAILED DESCRIPTION OF THE INVENTION

All disclosures of the Provisional Patent Application serial number 60/137,355 filed June 3, 1999, from which this application claims priority under 35 USC §119(e), are hereby incorporated by reference herein.

In an illustrative embodiment of the disclosed system, a network management database includes both 1) meta data, and 2) user data. Consistent with the present invention, a set of meta-schema entities are used to describe aspects of software objects that may be defined using the meta data, which is in turn used to generate the software objects of the user data. As

shown in Fig. 1, the illustrative embodiment of the disclosed system provides for installation of a network management database in 3 steps. At step 10, the meta-schema entities for describing and storing the meta-data

5   are installed. At step 12, the meta-schema entities loaded at step 10 are populated with meta-data entries reflecting, for example, a type of user application, such as a network management system. The meta-data loaded at step 12 describes the various types of objects

10   that are being operated on by the associated application program. For example, meta-data for a network management application may be used to describe properties of the various kinds of network devices that may be managed by the system.

15   Finally, at step 14, user data objects are generated for storing user data, and which reflect specific associated devices within a particular run-time environment. The user data objects loaded at step 14 are, for example, managed objects associated with

20   devices in a specific configuration of networked resources that are to be managed by a network management system. To facilitate convenient upgrades to a software application operating on the user data objects, each of the meta-data entities, as well as the user-data

25   entities, are "version tagged", such that entries within any of the meta-data entities or user-data entities are indexed in part based on a version tag identifying a specific software revision.

            Fig. 2 illustrates a number of meta-schema entities

30   used for describing and storing meta-data in the disclosed system. Specifically, several "entity" meta-schema entities 20, as well as several "relationship"

meta-schema entities 22 are provided, together with a versions table 24. The "entity" meta-schema entities 20 are shown including an object types entity 26, an attributes entity 28, an enumerated attribute values entity 30, and a conditions entity 32. The "relationship" meta-schema entities 22 are shown including a contain relationships entity 34, a consume relationships entity 36, and a connect relationships entity 38. Illustrative embodiments of the meta-schema entities shown in Fig. 2 are further described below.

Fig. 3 illustrates simultaneous inter-operation between a number of software objects 58 associated with a first revision 50 of a software program, and a number of software objects 60 associated with a second revision 54 of the software program. The revisions 50 and 54 of the software program are each shown associated with a respective version tag. Specifically, revision 50 is shown associated with a Version N version tag 52, and revision 56 is shown associated with a Version N+1 version tag 56. Through the use of the version tags 52 and 56, the disclosed system provides for coexistence between the first revision 50 and the second revision 54 of the software program.

Fig. 4 shows a table of column definitions for a meta-schema entity used to store information related to a number of versions of meta-data, corresponding to the meta-schema entity 24 of Fig. 2. Accordingly, each row in the table of column definitions shown in Fig. 4 fully specifies the attributes of a corresponding column in the meta-schema entity referred to as "md_version". The versions of meta-data are, for example, associated with respective software program revisions. As defined based

on the column definitions shown in Fig. 4, entries in
the resulting md_version table can be used to store
information describing all versions of metadata ever
introduced into the system. A version column 82 is
defined as the index for selecting entries in the
resulting table, such that the individual entries are
indexed by the value in the version column 82.
Moreover, as used herein, those column definitions
specified with "yes" in the Index Component column
attribute are generally the columns whose values are
combined to form an unique index identifying a
respective entry (or "row") within the table formed
based on a given table of column definitions.

In the md_version table, each entry also includes
information stored in the other columns 84, which, for
example, can be used to store information about the
installation state, installation status, time of
installation, software license and associated timestamp
of the corresponding program revision. Since meta-data
and user-data information is associated with version
numbers, the information in the md_version table can be
used to determine the version-relevance of specific
operations and/or data. In particular, as a given
installation goes through different stages, the
corresponding entry in the md_version table may be
updated accordingly, to reflect the current status of
the installation.

A table of column definitions is shown in Fig. 5
for an md_object table, which is the meta-schema entity
describing a number of meta-data object types, and which
corresponds to the meta-schema entity 26 shown in Fig.
2. Each entry (row) in the table of column definitions

-12-

shown in Fig. 5 defines a column for the resulting md_object table. Each entry (row) in the resulting md_object table formed based on the column definitions shown in Fig. 5 is a meta-data entity. For example, each entry in the md_object table, together with entries in the other illustrative tables making up the meta-schema entities shown in Fig. 2, may be used to store information regarding a number of object types defined as meta-data. Such meta-data object types may then be used to define user data entities, which may be used to store user data objects associated with managed devices in a network.

The index into entries within the md_object table defined by the column definitions shown in Fig. 5 includes the values of the version column 102 and name column 106. Use of the value of the version column 102 as a portion of the index for entries in the md_object table permits indexing by version number within object name. Such indexing permits each meta-data object type to take on different values for the other columns 104 across different revisions of a software program. Further, for purposes of example, the characteristics described in the other columns 104 of the md_object table reflect who may access, for example in terms of reading, writing, and or executing, specific meta-data object types. In addition, a usage column 106 indicates a usage status of an object type associated with an entry in the md_object table. The usage column 106 is used to "delete" an object type for a particular software revision by storing the 'NONE' usage value in that column for the corresponding entry in the md_object table. An usage column is similarly included in the

-13-

tables of column definitions for various other ones of
the meta-schema entities described below. Accordingly,
as meta-data evolves over time, information regarding
older meta-data object types, for example representing

5   older devices or classes of devices, need not actually
be deleted, but may instead be updated to indicate the
relationship, if any, of such older object types to
newer object types.

     Fig. 6 shows a table of column definitions for an

10   md_attribute table. Each row in the table of column
definitions shown in Fig. 6 specifies a column in the
md_attribute table. The md_attribute table formed based
on the column definitions shown in Fig. 6 is a meta-
schema entity having entries that each describe a non-

15   relational meta-data attribute. The md_attribute table
corresponds to the meta-schema entity 28 as shown in
Fig. 2. In the table of column definitions shown in
Fig. 6, a version column 122 permits indexing of
attributes based on meta-data versions associated with

20   respective software revisions. Further as shown in Fig.
6, for a given entry in the md_attribute table, the
value of the object_name column 124 stores the name of
an associated object type, while the value of the name
column 126 stores the name of the attribute itself. The

25   version 122, object_name 124, and name 126 columns are
used in combination as an index into the md_attribute
table. Thus, each entry in the md_attribute table,
defining a particular attribute, is indexed by a
combination of the values in those three columns. The

30   other columns 128 include a type column 130 which may
store a value (ENUM) indicating that an attribute has a
number of possible enumerated values found in the

enumerated attribute values meta-schema entity 30 shown
in Fig. 2. Similarly, the type column 130 may store a
value (ENUM_COND) indicating that certain values for an
attribute are associated with or correspond to an

5   indicated condition described in the conditions meta-
schema entity 32 shown in Fig. 2. The condition_name
column 132 may be used to store a name of a condition
described in the meta-schema entity 32 that is
associated with an attribute. The value of the version

10   column 122 is an index into the version column of the
md_version table defined by the table of column
definitions shown in Fig. 4, the value of the
object_name field 124 is an index into the name column
of the md_object table defined by the table of column

15   definitions shown in Fig. 5, and the value of the
condition_name column 132 is an index into the name
column of the md_condition table defined by the table of
column definitions shown in Fig. 8 below.

       Fig. 7 shows a table of column definitions for an

20   md_enum table. Accordingly, each row in the table of
column definitions shown in Fig. 7 specifies a column in
the md_enum table. The md_enum table stores a number of
entries corresponding to a number of respective meta-
data enumerated attribute values, and is an example of

25   the meta-schema entity 30 of Fig. 2. The table of
column definitions in Fig. 7 includes a number of
entries. The entries in the md_enum table are each
indexed by a combination of the values in the version
152, object_name 154, attribute_name 156 and name 157

30   columns. The entries in the md_enum table each describe
potential values for associated attributes. Thus
potential values for attributes can be defined for

specific attributes within specific objects within
specific versions. The other columns 158 of the md_enum
table include a condition name column 160, whose value
is the name of an associated condition further described
in the meta-schema entity 32, as shown in Fig. 2, and
which may affect, or be affected by, the associated
enumerated value. The value of the version column 152
is an index into the version column of the md_version
table defined by the table of column definitions shown
in Fig. 4, the value of the object_name column 154 is an
index into the name column of the md_object table
defined by the table of column definitions shown in Fig.
5. The value of the attribute_name column 156 is an
index into the name column of the md_attribute table
defined by the table of column definitions in Fig. 6,
and the value of the condition_name column 160 is an
index into the name column of the md_condition table
defined by the table of column definitions in Fig. 8
below.

Fig. 8 shows a table of column definitions for an
md_condition table. Accordingly, each row in the table
of column definitions shown in Fig. 8 specifies a column
in the md_condition table. The md_condition table
includes entries describing a number of meta-data
conditions, and corresponds to the meta-schema entity 32
of Fig 2. Each condition described by an entry in the
md_condition table is associated with a current state
representing the fact that an enumerated attribute value
of a particular object, at a certain level of the object
hierarchy, has taken on a particular enumerated value.
As shown in Fig. 8, the name column 184 stores the name
of the condition, and the   object_hname

-16-

column 186 stores a hierarchical name describing the associated object. As used herein, a "hierarchical name" of an object fully specifies that object within an object hierarchy, wherein the object hierarchy reflects parent-child relationships between objects. For example, if OBJECT_1 is a child of OBJECT_2, and OBJECT_3 is a parent of OBJECT_2, then the hierarchical name of OBJECT_1 would combine the object names OBJECT_3, OBJECT_2, and OBJECT_1 in a way that reflects that hierarchy. For example, a hierarchical name may be represented as a series of object names separated by "."s - "OBJECT_3.OBJECT_2.OBJECT_1" for the preceding example. Such a hierarchical name is applicable in any of the disclosed column definitions herein having names including the string "hname".

The enum_attr_name column 188 in Fig. 8 stores the name of the associated enumerated attribute for which the condition is applicable, and the enum_value column 190 indicates the value of the enumerated attribute that defines the condition. Accordingly, the version column 182 value is an index into the version column of the md_version table. The version column 182 and name column 184 values are combined to form an index indicating specific entries within the md_condition table.

The data structures defined in Figs. 9-14 are illustrative components within the "relational" meta-schema entities 22 shown in Fig. 2. With regard to Fig. 9, a table of column definitions for an md_id_attr table is shown. Accordingly, each row within the table of column definitions shown in Fig. 9 specifies a column in the md_id_attr table. The md_id_attr table includes a

number of entries, each of which describe a meta-data "identity" relational attribute. In a first illustrative embodiment, the md_id_attr table may be considered a portion of the attributes meta-schema entity 28 as shown in Fig. 2. However, since the attributes defined by the md_id_attr table are used within "contain" types of object relationships, the md_id_attr table may alternatively be considered to be part of the contain relationships meta-schema entity 34 of Fig. 2.

In an illustrative embodiment, a contain relationship requires both an entry in the md_id_attr able, and an entry in the md_contain table described in connection with Fig. 10 below. For a given identity attribute entry in the md_id_attr table, the value of the type column 202 indicates whether the relationship between a containing object and a contained object is "PRIMARY", indicating a direct hierarchical link between a parent and an immediate child object in the object hierarchy, or "SECONDARY", indicating that the link between the containing object and the contained object not direct, and accordingly may span several layers within the object hierarchy. The value of the auto_create column 204 describes the number of corresponding child objects to be auto-created for a parent object in response to the contain relationship. The value of the condition_name column 206 indicates the name of an associated condition, if applicable. The value of the version column 208 is an index into the version column of the md_version table, the value of the object_name column 210 is an index into the name column of the md_object table, the value of the id_name column

-18-

212 is an index into the id_name column of the md_contain table, and the value of the condition_name column 206 is an index into the name column of the md_condition table. The index for selecting entries in the md_id_attr table is a combination of the values in the version column 208, the object_name column 210, the name column 214, and the id_name column 212.

Each entry in the md_id_attr table stores information regarding a specific identity attribute. Each identity attribute describes a contained object that may be within one or more contain relationships. For a given entry in the md_id_attr table, the value of the object_name column 210 indicates the name of a contained object type within a contain relationship, and is an index into the md_object table. Such a contained object is considered a child object of the containing object. The value of the id_name column 212 for an entry in the md_id_attr table links the identity attribute for that entry to an entry in the md_contain table, and is an index into the id_name column of that table. The entry in the md_contain table identified by the value of the id_name column 212 identifies a parent resource object type (through the parent_hname column 234 value of that entry), such as an identifier resource, from which objects of the type indicated by the object_name column 210 may allocate unique identifiers. By defining the id_name column 212 as a component of the index used to select entries in the md_id_attr table, groups of contained child objects may conveniently be linked to a common containing parent object identifier resource.

-19-

Fig. 10 shows table of column definitions for an md_contain table. Accordingly, each row in the table of column definitions shown in Fig. 10 specifies a column in the md_contain table. The md_contain table is an

5   example of a meta-schema entity for storing meta-data regarding corresponding "contain" relationships. The md_contain table corresponds to the contain relationships meta-schema entity 34 of Fig. 2. Each entry in the md_contain table describes an associated

10   contain relationship. For a given entry, the md_contain table includes an id_name column 232 having a value linking the entry to one or more entries in the md_id_attr table. The value of the parent_hname column 234 for stores the hierarchical object name of a parent

15   object for one or more associated contain relationships, which is an identifier resource for the child objects of the contain relationships. The value of a min column 236 stores a minimum value of the unique identifier which is the resource allocated from the parent object

20   to the child object(s) within the contain relationship, and a max column 238 stores a value that is the maximum value for that unique identifier. The value of the version column 240 is an index into the version column of the md_version table defined in Fig. 4, and each

25   object name component in the parent_hname column 234 value is a key into the name column of the md_object table defined in Fig. 5. The combination of the version 240 and id_name 232 columns forms an index identifying the individual entries within the md_contain table.

30   In an illustrative embodiment, for two entries having column values {v1, i1, ph1} and {v2, i2, ph2} of columns version 240, id_name 232, and parent_hname 234,

SUBSTITUTE SHEET (RULE 26)

-20-

respectively, then if i1 equals i2, this fact implies that ph1 equals ph2. Accordingly, because identifier resources must be uniquely defined, no two different parent_hname values in simultaneously existing md_contain entries can be associated with the same id_name value. However, entries in the md_contain table having the same parent_hname value can be associated with multiple id_name values. Further in an illustrative embodiment, the parent_hname association of an id_name cannot be changed across different versions. Additionally, the same child object ('object_name' in an md_id_attr table entry) cannot be contained under the same parent object ('parent_hname' in the associated md_contain entry) via more than one id_name value or multiple md_id_attr entries. However, there can be multiple possible values for parent_hname for a particular child 'object_name' through different 'id_name' values and different 'md_id_attr' entries. These properties hold true across different versions. Identifier allocations and deallocations made in response to contain relationships may, for example, be performed via constructor functions associated with the relevant object types for each such contain relationship.

Fig. 11 shows a table of column definitions for an md_pool_attr table. Accordingly, each row in the table of column definitions shown in Fig. 11 specifies a column within the md_pool_attr table. The md_pool_attr table includes entries which describe pool attributes for use in consume relationships, together with entries in the md_consume table (see Fig. 12). Accordingly, in an illustrative embodiment, the md_consume table

corresponds to the meta-schema entity 36 of Fig. 2, and the md_pool_attr table may be considered a portion of either the meta-schema entity 36 or the meta-schema entity 28 of Fig. 2. As shown in Fig. 11, for a given

5   entry in the md_pool_attr table describing an associated pool resource, the value of the type column 252 indicates whether the type of the resource is either 'ACTUAL', meaning that the sum of all child object allocations for that resource should not exceed the

10   parent pool size, or 'PSEUDO', meaning that the individual allocation for each child object associated with the resource should not exceed a predetermined maximum. The value of the min column 254 for an entry in the md_pool_attr table defines a minimum value that a

15   child object must allocate from the resource pool, and the value of the max column 256 defines the maximum amount of the resource that a single child object can allocate. The value of the default column 258 indicates a default value that a child object allocates, and the

20   value of the condition_name column 260 specifies a name of any applicable condition associated with or affected by an entry. The value of the version column 262 is an index into the version column of the md_version table, the value of the object_name column 264 is an index into

25   the name column of the md_object table defined, the value of the pool_name column 266 is an index into the pool_name column of the md_consume table, and the value of the condition_name column 260 is an index into the name column of the md_condition table. In the

30   illustrative embodiment of Fig. 11, the combination of the values in the version 262, object_name 264, name 270, and pool_name 266 columns form the index into the

-22-

md_pool_attr table.   Including   the   value   of   the
pool_name column 266 within the index allows grouping of
multiple child objects sharing the same parent pool.

5   Fig. 12 shows a table of column definitions for an
md_consume table.   Accordingly, each row in the table of
column definitions shown in Fig. 12 specifies a column
within the md_consume table.   The md_consume table is
used to store a number of entries, each describing a
respective   meta-data   "consume"   relationship.   The
10   md_consume table corresponds to the meta-schema entity
36 of Fig. 2.   In each consume relationship, a child
object consumes resources from a pool of resources
associated with parent objects in the hierarchy of the
parent object indicated by the value of the parent_hname
15   column 302.   Each entry in the md_consume table is
linked to one or more entries in the md_pool_attr table
through the value of the pool_name column.   When such a
parent object is instantiated, its resource pool is
created with a size equal to the value stored in the
20   'md_size' column.   When the child object or objects are
instantiated, they consume portions of the resource pool
by allocations of a size between the 'min' and 'max'
column values in the associated 'md_pool_attr' entry,
after they are created.   Such child objects return the
25   portions of the pool resource that they allocate before
they are deleted.   Such allocations and deallocations
may be performed, for purposes of example, through use
of associated constructor and destructor functions
associated with the object type of such child objects.

30   In the case of a 'PSEUDO' consume relationship,
each child object is permitted to allocate no more of
the pool resource than the value of the 'md_size' column

associated with the parent object through the entry for the relationship in the md_consume table. Specifically, the value of the pool_name column 306 for a given relationship is the name of the pool resource, and accordingly may, for example, be an index into a table defining a number of resource pools. The value of the parent_hname column for a given relationship entry in the md_consume table is a '.' delimited, hierarchical object name, having constituent object names separated by '.'s. As mentioned above, hierarchical object names fully specify an object within the object hierarchy, by including a string of parent object names for respective object hierarchy levels, and leading to the root object of the object hierarchy. In an illustrative embodiment, a child object of a given consume relationship is permitted to consume a pool resource associated with any object identified in the value of the parent_hname column 302 for that relationship. Accordingly, if the value of the parent_hname column 302 includes the hierarchical name of an object as a substring, then the child object of that consume relationship may allocate from a pool resource associated with that object.

Further as shown in Fig. 12, the value of the parent_attr_name column 308 indicates the name of the pool resource attribute associated with the parent object that is fully specified by the value of the parent_hname column 302. Accordingly, for a given consume relationship entry in the md_consume table, the value of the version column 310 is an index into the version column of the md_version table, the value of the pool_name column 306 is an index into a pool name column of an md_pool_attr table describing various pool

-24-

resources, the hierarchical object name stored in the parent_hname column 302 is an index into the name column of one or more entries in the md_object table defined, and the value of the parent_attr_name column 308 is an index into the name column of the md_attribute table. The index for entries in the md_consume table is, for example, the combination of the values in the version column 310 and the pool_name column 306.   In an illustrative embodiment, a resource pool is uniquely identified by the values in the parent_hname and parent_attr_name columns 302 and 308, and such a pool identity cannot be changed across different versions.

Fig. 13 shows a table of column definitions for an md_ptr_attr table. Accordingly, each row in the table of column definitions shown in Fig. 13 specifies a column in the md_ptr_attr table. The disclosed connect relationships each require an entry in the md_ptr_attr table, which describes a pointer attribute, together with an associated entry in the md_connect table (see Fig. 14). Each entry in the md_ptr_attr table describes a pointer relational attribute. Each entry in the md_ptr_attr table includes an object_name column 352, for storing the name of a source object of an associated connect relationship. The value of the type column 354 indicates whether the associated connect relationship is permanent or transient. If the connect relationship is permanent, then the endpoint object of the connect relationship cannot be changed during life time of the source object.   If the connect relationship is transient, then the endpoint object can be dynamically redefined. The value of the min column 356 indicates the minimum pointer value an endpoint object of the

-25-

associated connect relationship must allocate. The value of the max column 358 indicates the maximum pointer value that an endpoint object can allocate within a given connect relationship. The value of the

5   condition_name column 360 indicates the name of any associated condition, if applicable. Additionally, for a given connect relationship, the value of the version column 362 is a key into the version column of the md_version table defined in Fig. 4, and the value of the

10   object_name column 352 defines the endpoint object of the relationships and is an index into the name column of the md_object table defined in Fig. 5. The value of the ptr_name column 362 is an index into the ptr_name column of the md_connect table defined in Fig. 14, and

15   therefore links an entry within the md_ptr_attr table and an entry within the md_connect table, thus associating the two tables to form respective connect relationships from associated entries within the tables. The value of the condition name column 360 is an index

20   into the name column of the md_condition table defined in Fig. 8. In the illustrative embodiment of Fig 13, the index for selecting entries within the md_ptr_attr table is the combination of the version 362, object_name 352, and name 364 columns. Because the value of the

25   ptr_name column 362 is not included in the primary key, different pointer relational attributes cannot share the same group of destinations.

Fig. 14 shows a table of column definitions for an md_connect table. The md_connect table is an example of

30   a meta-schema entity for describing a number of meta-data "connection" relationships. Entries in the md_connect table define a number of corresponding

-26-

'connect' relationships between objects. The connect relationships may be used to connect a source object to one or more endpoint objects. For a given connect relationship entry in the md_connect table, a value of the ptr_name column 402 connects the entry to a corresponding entry in the md_ptr_attr table, which defines the source object for the connect relationship in its object_name column value. The value of the des_hname column 404 defines the hierarchical object name for the destination object or objects of the connect relationship. The value of the des_scope_hname column 406 defines a starting point within the object hierarchy for choosing a possible destination or destinations within the associated connect relationship. The value of the des_share_hname column 408 indicates a common parent object within the object hierarchy for all destination objects of a relationship, and the value of the des_max_ref_cnt column defines the maximum number of times a particular destination object instance can be referenced by a particular pointer indicated by the value in the ptr_name column 402. In this way, a single source object can point to multiple destination objects. For a given entry in the md_connect table, the value of the version column 412 is an index into the version column of the md_version table, the value of the ptr_name column is an index into the name column of an md_ptr_attr table describing a number of system wide resources. In the illustrative embodiment, the combination of the values in the version, 412, ptr_name 402 and des_hname 404 columns defines the index for identifying individual entries, corresponding to connection relationships, in the md_connect table.

-27-

Figs. 15-26 show user data objects corresponding to the user schema loaded at step 14 of Fig. 1. The illustrative user data objects in Figs. 15-26 describe various actual devices that are managed by a network management system, and are formed in response to the meta-data entered into the meta-schema entities shown in Fig. 2. In the illustrative embodiment, user data consists of the following tables:

a) ud_user

b) a ud_{md_object.name} table for every row in 'md_object'

c) ud_contain

d) ud_consume

e) ud_connect

After the successful generation of the user data tables for a given revision of the software program operating on them, the status of the entry for that revision in the 'md_version' table is modified to reflect the successful installation of the user data.

Fig. 15 shows a table of column definitions for a ud_user table. The ud_user table includes entries that describe various users of the system. For a given entry in the ud_user table, associated with a respective user, the value of the version column 452 defines a software revision with which the user is associated, the value of the name column 454 defines the name of the user, the value of the md_group column 456 defines a group of users with which the user is associated, the value of the mask column 458 defines a default Unix access mask (e.g. "rwxr-xr-x") associated with the user, the value

-28-

of the password column 460 is a password associated with the user, and the value of the expire_timestamp column 462 defines an optional expiration date for the user. The value of the version column 452 is an index into the version column of the md_version table shown in Fig. 4. Each entry in the ud_user table is indexed by a value contained in the combined version 452 and name 454 columns. A number of default entries 464 for the ud_user table are also shown in Fig. 15.

Fig. 16 shows an example of an md_object table formed based on the table of column definitions in Fig. 5, and loaded with a number of meta-data entries 502. As noted above, there is one ud_{md_object.name} created for every row in the md_object table. Accordingly, for every one of the entries 502 in the illustrative md_object table 500, a corresponding user data table will be created, resulting, for example, in the following device tables:

ud_Root

ud_SN6000

ud_SN8000

ud_SN8001

ud_SN8600

ud_SN8400

ud_SN2000

The ud_Root table functions as a placeholder in the object hierarchy of the device tables, while each of the entries in the ud_SN6000, ud_SN8000, ud_SN8001, ud_SN8600, ud_SN8400, and ud_SN2000 device tables are associated with corresponding physical devices of the

-29-

device type associated with the respective device table. Each entry in one of the device tables may also be referred to as a "device instance" or "device entry". For example, each entry in the ud_SN600 table describes a device of the SN_6000 device type, each entry in the ud_SN8000 table describes a device of the SN_8000 type, and so forth. In the illustrative embodiment, each of these user tables are generated with the default columns 550 specified in the table of column definitions shown in Fig. 17. The default columns 550 include a version column 552 indicating a software revision with which a device entry ("instance") is associated, an owner column 554 indicating a user who created the device entry, an md_group column 556, and a unique identifier column 558 containing a unique identifier value associated with the device entry. A number of access related columns 560 are further provided to describe access privileges associated with each device entry.

For every meta-data row within the md_attribute table, a corresponding column will be included in each of the device tables. For example, if the md_attribute table has the two entries 582 and 584 shown in the illustrative md_attribute table 580 of Fig. 18, then, for example, tables ud_SN6000 and ud_SN8000 602 will be formed having columns specified by the tables of column definitions 600 and 602 shown in Fig. 19. As specified by the table of column definitions 600 in Fig. 19, the ud_SN6000 device table includes default columns 604, as well as additional columns 606 reflecting the md_attribute table 580 in Fig. 18. Similarly, the ud_SN8000 device table 602 includes default columns 608,

-30-

as well as additional columns 610 reflecting the md_attribute table 580 in Fig. 18.

An illustrative configuration of the md_id_attr table defined in Fig. 9 is shown as md_pool_attr table 650 in Fig. 20. For every row in the md_id_attr table 650, a corresponding column of the name [name]$[id_name] will be added to device table 'ud_[object_name]'. For example, if the md_id_attr table has the meta-data entries 652 of the illustrative md_id_attr table 650 shown in Fig. 20, then the device tables ud_SN8001 and ud_PowerSupply will be generated with the extra columns 704 specified for ud_SN8001 and extra columns 706 specified for ud_PowerSupply 702 in Fig. 21.

An illustrative configuration of the md_pool_attr table specified in Fig. 11 is illustrated in Fig. 22 by the table md_pool_attr 750. For every row in the md_pool_attr table 750, a corresponding column of the name [name]#[pool_name], using the # character vs. the '$' character to distinguish over the above described [name]$[id_name] format, will be added to the ud_[object_name] table. For example, if the md_pool_attr table has the entries 752 shown in the illustrative md_pool_attr table 750 of Fig. 22, then the device tables ud_PowerSupply and ud_OC12Port would have the extra columns 804 and 806 respectively, in addition to the default columns 812 and 810, as shown by the tables of column definitions shown in Fig. 23.

Fig. 24 shows a table of column definitions for a ud_contain table. Accordingly, each row in the table of column definitions shown in Fig. 24 specifies a column in the ud_contain table. The ud_contain table specified by the table of column definitions defined in Fig. 24

stores a number of entries, each one or which represents a contain relationship instance between instances of managed objects. Thus, each entry in the md_contain table represents an instantiation of a contain relationship. Each contain relationship instantiation is associated with an entry in the md_id_attr table and an entry in the md_contain table. For a given entry in the ud_contain table, the values of various columns are now described. The value of the version column 902 is an index into the version column of the md_version table, and contains the same value as in the version columns of the associated entries in the md_contain and md_id_attr tables. The value of the child_name column 904 is a key into the object_name column of the md_contain table, and is equal to the name of the contained object type, as also indicated by the value in the object_name column value of the associated entry in the md_id_attr table. The value of the attribute_name column 906 is the name of the identifier attribute for the associated contain relationship, and is equal to the value of the name column in the associated entry in the md_id_attr table. The value of the id_name column 908 is an index into the id_name column of the md_contain table, as well as the id_name column of the md_id_attr table, and therefore associates entries in the ud_contain, md_contain, and md_id_attr tables that together define an instance of a connect relationship. The value of the id_type field for an entry in the ud_contain table is equal to the type field of the associated entry in the md_id_attr table. The value of the parent_hname field 912 for an entry in the ud_contain table is equal to the value in the

parent_hname field of the associated entry in the md_contain table. The value of the child_pkey column 913 for an entry in the ud_contain table is a unique identifier for the instance of the contained object of the associated contain relationship instance. The value of the child_id column 915 is a an identifier for the contained object instance in the associated contain relationship instance, and is specifically the identifier allocated for the contained object instance from the identifier resource associated with the contain relationship instance. The value of the parent_hpid column 916 for a given entry in the ud_contain table is an identifier which fully specifies the parent object instance for the associated contain relationship instance within the object instance hierarchy, reflecting the hierarchical structure of parent object instances at higher levels of the object instance hierarchy from the contained object instance, thus permitting traversal of the object instance hierarchy from the child object instance of the associated contain relationship instance. The value of the parent_hpkey column 912 for an entry within the ud_contain table is a unique identifier for the parent object instance of the associated contain relationship instance. The value of the expire_timestamp column for a given entry in the ud_contain table stores an expiration time for the associated contain relationship instance. In this way, identifier relationships associated with contain relationships can be made for fixed time periods, after which the identifier for the relationship is returned unless further action is taken to reset the expiration time for the relationship.

The table of column definitions shown in Fig. 25 defines the columns for the ud_consume table. Accordingly, each row in the table of column definitions shown in Fig. 25 specifies a column in the ud_consume table. The entries in the ud_consume table are each associated with a respective entry in the md_pool_attr table and a respective entry in the md_consume table. Together, these associated entries specify a consume relationship instance between instances of managed objects. Values for a given entry in the ud_consume table are now described. The value of the version column 952 is equal to the value of the version columns in the associated entries in the md_consume and md_pool_attr tables. The value of the child_name column 954 is equal to the value of the object_name column in the associated entry in the md_pool_attr table. The value of the attribute_name column 956 is the same as the name column in the associated entry in the md_pool_attr table. The value of the pool_name column 958 is equal to the value of the pool_name column of the associated entry in the md_pool_attr table and the associated entry in the md_consume table, thus connecting the relevant entries in these three tables related to the consume relationship instance. The value of the pool_type 952 column is equal to the value of the type column in the associated entry in the md_pool_attr table. The value of the parent_hname 960 column is equal to the value of the parent_hname column in the associated entry in the md_consume table. The value of the parent_attr_name 962 is the value of the name column for the associated entry in the md_pool_attr table. The value of the child_pkey column 963 is a unique

identifier for the instance of the child object within
the consume relationship instance. The value of the
child pool column 965 is the amount of the consumed
resource currently allocated by the child object

5   instance of the consume relationship instance. The
value of the parent_hpkey column 964 is a unique
identifier of the instance of the parent object for the
associated consume relationship instance. The value of
the expire_timestamp column 966 is an expiration time

10   associated with the consume relationship instance for
the entry.

       Fig. 26 shows a table of column definitions for a
ud_connect table. Accordingly, each row in the table of
column definitions shown in Fig. 26 specifies a column

15   within the ud_connect table. The ud_connect table
stores a number of user data entries that, together with
associated entries in the md_connect and md_ptr_attr
tables, define instances of connect relationships
between user data instances of managed objects. For a

20   given entry in the ud_connect table associated with a
particular connect relationship instance, a number of
column values are now described. The value of the
version column is 1002 is the same as the version column
values in the associated md_connect and md_ptr_attr

25   table entries. The value of the src_name column 1004 is
equal to the object_name column value in the associated
entry within the md_ptr_attr table. The value of the
attribute_name column 1006 is the same as the value of
the name column of the associated entry in the

30   md_ptr_attr table. The value of the ptr_name column
1008 is the same as value of the ptr_name column of the
associated entry in the md_ptr_attr table. The value of

the ptr_type column 1009 is the same as the value of the type column in the associated entry in the md_ptr_attr table. The value of the des_hname column 1009 is the same as the value in the des_hname column of the associated entry in the md_connect table. The value of the src_pkey column 1011 is a unique identifier for the instance of the source object of the associated connect relationship instance. The value of the des_pkey column 1013 is a unique identifier for the instance of the destination object of the associated connect relationship instance. The value of the des_hpkey column 1012 is a hierarchical pkey (primary key) of the instance of the destination object for the associated connect relationship, and the value of the des_order column 1014 reflects an ordering between common destinations of a single source. The value of the expire_timestamp column 1015 indicates a time at which the associated relationship instance expires.

Those skilled in the art should readily appreciate that the programs defining the functions of the present invention can be delivered to a computer in many forms; including, but not limited to: (a) information permanently stored on non-writable storage media (e.g. read only memory devices within a computer such as ROM or CD-ROM disks readable by a computer I/O attachment); (b) information alterably stored on writable storage media (e.g. floppy disks and hard drives); or (c) information conveyed to a computer through communication media for example using baseband signaling or broadband signaling techniques, including carrier wave signaling techniques, such as over computer or telephone networks via a modem. In addition, while the invention may be

-36-

embodied in computer software, the functions necessary to implement the invention may alternatively be embodied in part or in whole using hardware components such as Application Specific Integrated Circuits or other hardware, or some combination of hardware components and software.

While the invention is described through the above exemplary embodiments, it will be understood by those of ordinary skill in the art that modification to and variation of the illustrated embodiments may be made without departing from the inventive concepts herein disclosed. Moreover, while the preferred embodiments are described in connection with various illustrative data structures, one skilled in the art will recognize that the system may be embodied using a variety of specific data structures. Accordingly, the invention should not be viewed as limited except by the scope and spirit of the appended claims.

-37-

CLAIMS

What is claimed is:

5   1.   A system for supporting revisions to a software
           program, comprising;
              at least one data structure of a first type, having
           at least one element, said at least one element of said
           data structure of said first type describing at least
10   one relationship between a first object type and a
           second object type;
              at least one data structure of a second type,
           having at least one element, said at least one element
           in said data structure of said second type describing at
15   least one entity of said first object type; and
              wherein each said element in said data structure of
           said first type is associated with a respective revision
           of said software program.

20   2. The system of claim 1, wherein each said element in
         said data structure of said second type is associated
         with a respective revision of said software program.

     3. The system of claim 1, wherein each said respective
25   revision of said software program is one of a plurality
         of revisions of said software program.

     4. The system of claim 1, wherein said at least one
         relationship has a relationship type, wherein said
30   relationship type is one of a plurality of relationship
         types, wherein said plurality of relationship types

SUBSTITUTE SHEET (RULE 26)

-38-

comprise a contain relationship type, a consume
relationship type, and a connect relationship type.

5. The system of claim 4, wherein said relationship type
of said at least one relationship is said contain
relationship type, wherein said contain relationship
type indicates that a plurality of object instances of
said second object type are instantiated responsive to a
resource of an identified object instance of said first
type, and wherein said resource comprises a plurality of
unique identifiers, each one of said plurality of unique
identifiers associable with a respective one said object
instances of said second object type.

6. The system of claim 5, further comprising a
constructor function associated with said second object
type, wherein said constructor function requests one of
said plurality of unique identifiers from said
identified object instance of said first object type,
and wherein said constructor function fails to create an
object instance of said second object type in the event
that none of said plurality of unique identifiers are
available.

7. The system of claim 6, wherein said resource
comprises a counter, wherein said counter is incremented
for each of said plurality of object instances of said
second object type, and wherein said plurality of unique
identifiers comprise respective values of said counter.

-39-

8. The system of claim 7, wherein none of said plurality of unique identifiers are available when said counter reaches a predetermined maximum value.

9. The system of claim 8, wherein said identified object instance of said first object type uses said plurality of unique identifiers to uniquely identify each of said plurality of object instances of said second object type.

10. The system of claim 9, further comprising a destructor function associated with said indicated object instance of said first object type, wherein said destructor function deletes each of said plurality of object instances of said second object type responsive to a request for deletion of said object instance of said first object type.

11. The system of claim 4, wherein said relationship type of said at least one relationship is said consume relationship type, wherein said consume relationship type indicates that a plurality of object instances of said second object type are instantiated responsive to at least one resource of an indicated object instance of said first object type, and wherein said resource comprises a variably allocatable pool, and wherein a first one of said plurality of object instances of said second object type consumes a different amount of said variably allocatable pool than a second one of said plurality of object instances of said second object type.

-40-

12. The system of claim 11, wherein said variably
allocatable resource represents communication bandwidth.

13. The system of claim 11, further comprising a
constructor function associated with said second object
type, wherein said constructor function requests a
portion of said variably allocatable pool from said
identified object instance of said first object type,
and wherein said constructor function fails to create an
object instance of said second object type in the event
that a requested amount said variably allocatable pool
exceeds an available amount of said variably allocatable
resource.

14. The system of claim 13, wherein said variably
allocatable pool is represented by a sum of current
allocations from said variably allocatable pool.

15. The system of claim 14, further comprising a
destructor function associated with said indicated
object instance of said first object type, wherein said
destructor function deletes each of said plurality of
object instances of said second object type responsive
to deletion of said object instance of said first object
type.

16. The system of claim 4, wherein said relationship
type of said at least one relationship is said connect
relationship type, wherein said connect relationship
type indicates that each of a plurality of data objects
of said second object type is instantiated in

association with a respective data object of said first
type.

1/25

INSTALL META SCHEMA ⟋10

POPULATE METADATA ⟋12

INSTALL USER SCHEMA ⟋14

FIG. 1

FIG. 2

VERSIONS ⟋24

OBJECT TYPES 26

ATTRIBUTES 28

30 ENUMERATED ATTRIBUTE VALUES

CONDITIONS 32

CONTAIN RELATIONSHIPS
34

20

CONSUME RELATIONSHIPS 36

CONNECT RELATIONSHIPS 38

22
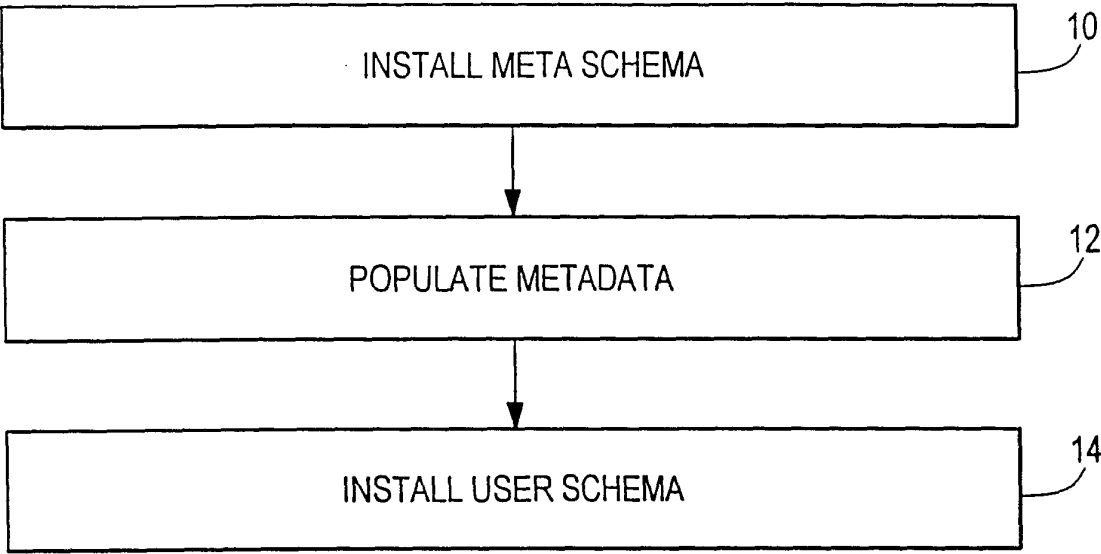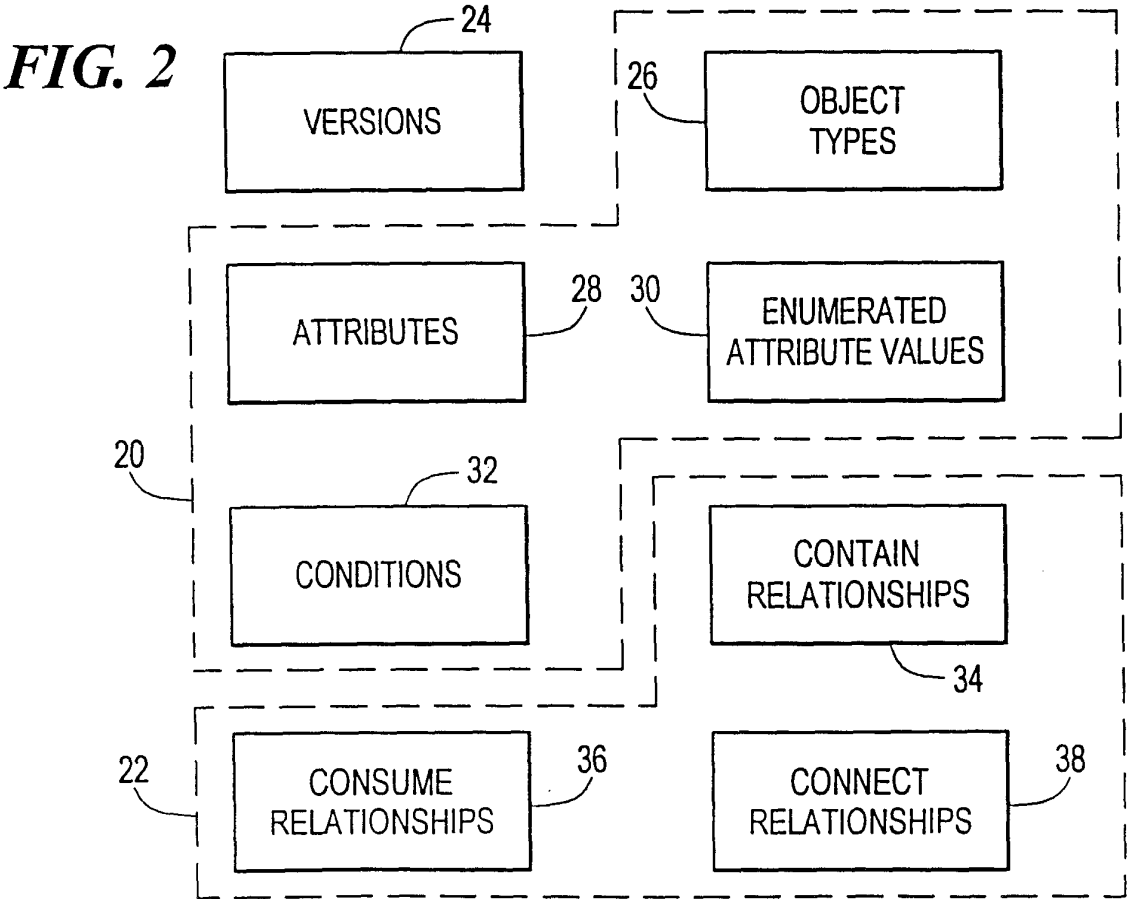
2/25
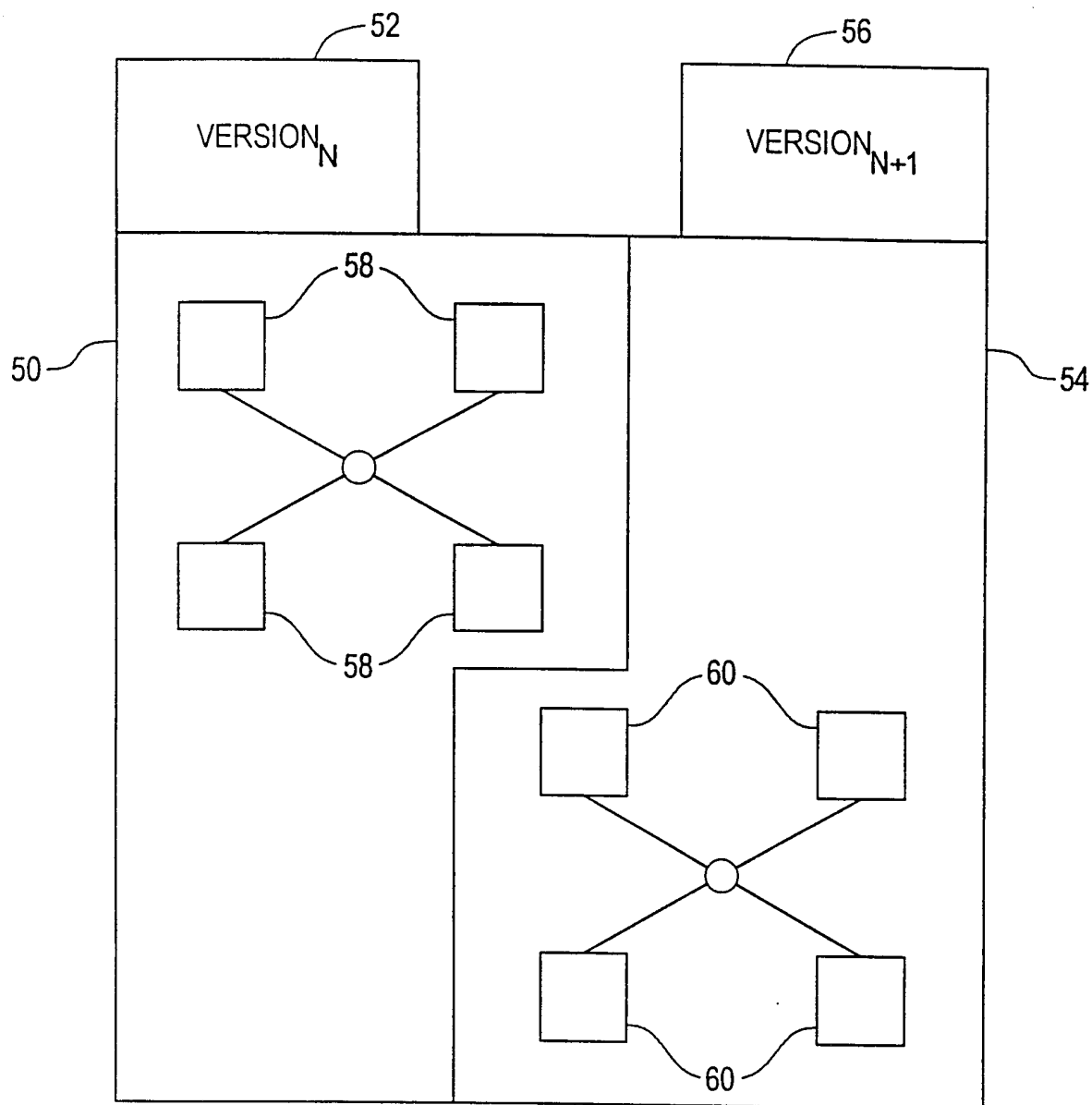
FIG. 3

name	type	length	null	index component	choices 123456789 123456
version	String	16	no	yes	-
install_state	String	16	no	no	metaschema metadata inserted installed failed_install uninstalled failed_uninstall
install_status	String	16	no	no	
install_timestamp	Date	-	no	no	-
license	String	16	no	no	permanent timed expired retired
license_timestamp	Date	-	no	no	-

82

84

COLUMN DEFINITIONS
FOR md_version

FIG. 4

columns:

name	type	length	null	index component	choices 123456789 123456
version	String	16	no	yes	-
name	String	32	no	yes	-
alias	String	50	no	no	-
description	String	128	yes	no	-
owner	String	16	no	no	-
md_group	String	16	no	no	-
owner_access	String	3	no	no	-
group_access	String	3	no	no	-
other_access	String	3	no	no	-
usage	String	16	no	no	DB_ONLY NET_ONLY DB_NET NONE
flags	String	128	yes	no	-

102

104

106

COLUMN DEFINITIONS
FOR md_object

FIG. 5

columns:

name	type	length	null	index component	choices 123456789 123456
version	String	16	no	yes	-
object_name	String	32	no	yes	-
name	String	32	no	yes	-
alias	String	50	no	no	-
description	String	128	yes	no	-
owner	String	16	no	no	-
md_group	String	16	no	no	-
owner_access	String	3	no	no	-
group_access	String	3	no	no	-
other_access	String	3	no	no	-
usage	String	16	no	no	DB_ONLY NET_ONLY DB_ONLY NONE
flags	String	128	yes	no	-
type	String	16	no	no	STRING NUMBER DATE ENUM ENUM_COND NUMBER_POOL
instance_label	String	16	yes	no	-
md_size	Number	4	-	no	-
unit	String	16	yes	no	-
null_val	String	8	no	no	YES NO
range	String	256	yes	no	-
default_val	String	64	yes	no	null, " ", or other non-null string
condition_name	String	32	yes	no	-

122   124   126   130   128   132

COLUMN DEFINITIONS
FOR md_attributes

FIG. 6

columns:

name	type	length	null	index component	choices 123456789 123456
version	String	16	no	yes	-
object_name	String	32	no	yes	-
attribute_name	String	32	no	yes	-
name	String	32	no	yes	-
alias	String	50	no	no	-
description	String	128	yes	no	-
owner	String	16	no	no	-
md_group	String	16	no	no	-
owner_access	String	3	no	no	-
group_access	String	3	no	no	-
other_access	String	3	no	no	-
usage	String	16	no	no	DB_ONLY NET_ONLY DB_NET NONE
value	Number	16	-	no	-
condition_name	String	32	yes	no	-

152 154 156 157 — 158 — 160

COLUMN DEFINITIONS FOR md_enum

FIG. 7

columns:

name	type	length	null	index component	choices 123456789 123456
version (182)	String	16	no	yes	-
name (184)	String	32	no	yes	-
object_hname (186)	String	32	no	no	-
enum_attr_name (188)	String	32	no	no	-
enum_value (190)	String	32	no	no	-

COLUMN DEFINITIONS
FOR md_condition

FIG. 8

columns:

name	type	length	null	index component	· choices 123456789 123456
version ———— 208	String	16	no	yes	-
object_name — 210	String	32	no	yes	-
name ———— 214	String	32	no	yes	-
id_name — 212	String	32	no	yes	-
alias	String	50	no	no	-
description	String	128	yes	no	-
owner	String	16	no	no	-
md_group	String	16	no	no	-
owner_access	String	3	no	no	-
group_access	String	3	no	no	-
other_access	String	3	no	no	-
usage	String	16	no	no	DB_ONLY NET_ONLY DB_NET NONE
flags	String	128	yes	no	-
type — 202	String	16	no	no	PRIMARY SECONDARY
instance_label	String	16	yes	no	-
auto_create — 204	Number	16	-	no	-
condition_name — 206	String	32	yes	no	-

COLUMN DEFINITIONS
FOR md_id_attr

FIG. 9

columns:

name	type	length	null	index component	choices 123456789 123456
version	String	16	no	yes	-
id_name	String	32	no	yes	-
parent_hname	String	527	no	no	-
min	Number	16	-	no	-
max	Number	16	-	no	-

240 — version
232 — id_name

234 — parent_hname
236 — min
238 — max

COLUMN DEFINITIONS
FOR md_contain

FIG. 10

columns:

name	type	length	null	index component	choices 123456789 123456
version — 262	String	16	no	yes	-
object_name 264	String	32	no	yes	-
name — 270	String	32	no	yes	-
pool_name 266	String	32	no	yes	-
alias	String	50	no	no	-
description	String	128	yes	no	-
owner	String	16	no	no	-
md_group	String	16	no	no	-
owner_access	String	3	no	no	-
group_access	String	3	no	no	-
other_access	String	3	no	no	-
usage	String	16	no	no	DB_ONLY NET_ONLY DB_NET NONE
flags	String	128	yes	no	-
type 252	String	16	no	no	ACTUAL PSEUDO
instance_label	String	16	yes	no	-
min 254	Number	16	-	no	-
max 256	Number	16	-	no	-
default 258	Number	16	-	no	-
condition_name 260	String	32	yes	no	-

COLUMN DEFINITIONS
FOR md_pool_attr

FIG. 11

columns:

name	type	length	null	index component	choices 123456789 123456
version	String	16	no	yes	-
pool_name	String	32	no	yes	-
parent_hname	String	527	no	no	-
parent_attr_name	String	32	no	no	-
md_size	Number	16	.	no	-

310 version
306 pool_name
302 parent_hname
308 parent_attr_name
304 md_size

COLUMN DEFINITIONS
FOR md_consume

FIG. 12

columns:

name	type	length	null	index component	choices 123456789 123456
version — 362	String	16	no	yes	-
object_name — 352	String	32	no	yes	-
name — 364	String	32	no	yes	-
ptr_name — 362	String	32	no	no	-
alias	String	50	no	no	-
description	String	128	yes	no	-
owner	String	16	no	no	-
md_group	String	16	no	no	-
owner_access	String	3	no	no	-
group_access	String	3	no	no	-
other_access	String	3	no	no	-
usage	String	16	no	no	DB_ONLY NET_ONLY DB_NET NONE
flags	String	128	yes	no	-
type — 354	String	16	no	no	PERMANENT TRANSIENT
min — 356	Number	16	-	no	-
max — 358	Number	16	-	no	-
condition_name — 360	String	32	yes	no	-

COLUMN DEFINITIONS
FOR md_ptr_attr

FIG. 13

columns:

name	type	length	null	index component	choices 123456789 123456
version	String	16	no	yes	-
pt_name	String	32	no	yes	-
des_hname	String	527	no	yes	-
des_scope_hname	String	527	no	no	-
des_share_hname	String	527	no	no	-
des_max_ref_cnt	Number	16	-	no	-

412 — version
402 — pt_name
404 — des_hname
406 — des_scope_hname
408 — des_share_hname
410 — des_max_ref_cnt

COLUMN DEFINITIONS
FOR md_connect

FIG. 14

name	type	length	null	index component	choices
version	String	16	no	yes	123456789 123456
name	String	16	no	yes	-
md_group	String	16	no	no	-
mask	String	10	no	no	-
password	Number	16	no	no	-
expire_timestamp	Date	-	yes	no	-

452 — version
454 — name
456 — md_group
458 — mask
460 — password
462 — expire_timestamp

default entries in the tables are: 464 →

version	name	md_group	mask	password	date
00.00.01	root	root	rwxr-xr-x	108698511	:
00.00.01	admin	system	rwxrwxr-x	-1422235966	:
00.00.01	operator	staff	rwxrwxr-x	1662708749	:
00.00.01	guest	guest	r-r-r-	0	:

COLUMN DEFINITIONS
FOR ud_user

FIG. 15

md_object

version	name	alias	desc	⋮
00.00.01	Root	⋮	⋮	⋮
00.00.01	SN6000	⋮	⋮	⋮
00.00.01	SN8000	⋮	⋮	⋮
00.00.01	SN8001	⋮	⋮	⋮
00.00.01	SN8600	⋮	⋮	⋮
00.00.01	SN8400	⋮	⋮	⋮
00.00.01	SN2000	⋮	⋮	⋮

500

502

FIG. 16

16/25

name	type	length	null	no	choices 123456789 123456
version ——552	String	16	no	no	.
owner ——554	String	16	no	no	.
md_group ——556	String	16	no	no	.
owner_access	String	3	no	no	.
group_access ——560	String	3	no	no	.
other_access	String	3	no	no	.
flags	String	128	yes	no	.
unique_id ——558	Number	16	-	yes	.
pid_name	String	65	no	no	.

Default Columns — 550

FIG. 17

md_attribute

580 →

version	object_name	name	...	type	size	null_val	...
00.00.01	SN6000	name	...	STRING	64	no	...
00.00.01	SN6000	location	...	STRING	64	no	...
00.00.01	SN6000	ip_addr	...	STRING	15	no	...
00.00.01	SN8000	name	...	STRING	64	no	...
00.00.01	SN8000	location	...	STRING	64	no	...
00.00.01	SN8000	ip_addr	...	STRING	15	no	...
00.00.01	SN8000	shelf_id	...	NUMBER	16	no	...

582

584

FIG. 18

COLUMN DEFINITIONS
FOR   600
ud_SN6000:

name	type	length	null	default_val

604 {
version	...			
owner	...			
md_group	...			
owner_access	...			
group_access	...			
other_access	...			
flags	...			
unique_id	...			
pid_name	...			

606 {
name	String	64	no	..
location	String	64	no	..
ip_addr	String	15	no	..

COLUMN DEFINITIONS
FOR   602
ud_SN8000:

name	type	length	null	default_val

608 {
version	...			
owner	...			
md_group	...			
owner_access	...			
group_access	...			
other_access	...			
flags	...			
unique_id	...			
pid_name	...			

610 {
name	String	64	no	..
location	String	64	no	..
ip_addr	String	15	no	..
shelf_id	Number	16	no	..

FIG. 19

md_id_attr

650 →

version	object_name	name	id_name		type	
00.00.01	SN8001	shelf_id	ExpShelves	⋮	PRIMARY	⋮
00.00.01	SN8001	node-id	RootNes	⋮	SECONDARY	⋮
00.00.01	PowerSupply	ps_id	IrPsSlots	⋮	PRIMARY	⋮
00.00.01	PowerSupply	ps_id	RedPsSlots	⋮	PRIMARY	⋮
00.00.01	PowerSupply	ps_id	RedExpPsSlots	⋮	PRIMARY	⋮

652

FIG. 20

COLUMN DEFINITIONS
FOR　　　　　　　　　　700

ud_SN8001:

name	type	length	null	default_val
version	. . .			
owner	. . .			
md_group	. . .			
owner_access	. . .			
group_access	. . .			
other_access	. . .			
flags	. . .			
unique_id	. . .			
pid_name	. . .			
. . .	. . .			
shelf_id$ExpShelves	Number	16	yes	null
node_id$RootNes	Number	16	yes	null

704 {

COLUMN DEFINITIONS
FOR　　　　　　　　　　702

ud_PowerSupply:

name	type	length	null	default_val
version	. . .			
owner	. . .			
md_group	. . .			
owner_access	. . .			
group_access	. . .			
other_access	. . .			
flags	. . .			
unique_id	. . .			
pid_name	. . .			
. . .	. . .			
ps_id$lrPsSlots	Number	16	yes	null
ps_id$RedPsSlots	Number	16	yes	null
ps_id$RedexpPsSlots	Number	16	yes	null

706 {

FIG. 21

md_pos_attr

750 →

version	object_name	name	pool_name		type	
00.00.01	PowerSupply	current	IrPsCurrent	⋮	PSEUDO	⋮
00.00.01	PowerSupply	current	RedPsCurrent	⋮	PSEUDO	⋮
00.00.01	PowerSupply	current	RedExpPsCurrent	⋮	PSEUDO	⋮
00.00.01	OC12Port	port_bw	RedWic40C12Bw	⋮	ACTUAL	⋮
00.00.01	OC12Port	port_bw	ERedWic40C12Bw	⋮	ACTUAL	⋮

752

FIG. 22

22/25

COLUMN DEFINITIONS
FOR   800
ud_PowerSupply:

name	type	length	null	default_val
version	. . .			
owner	. . .			
md_group	. . .			
owner_access	. . .			
group_access	. . .			
other_access	. . .			
flags	. . .			
unique_id	. . .			
pid_name	. . .			
. . .	. . .			
current#IrPsCurrent	String	16	yes	null
current#RedPsCurrent	String	16	yes	null
current#RedExpPsCurrent	String	16	yes	null

812 { version ... pid_name }

804 { current#IrPsCurrent ... current#RedExpPsCurrent }

COLUMN DEFINITIONS
FOR   802
ud_OC12Port:

name	type	length	null	default_val
version	. . .			
owner	. . .			
md_group	. . .			
owner_access	. . .			
group_access	. . .			
other_access	. . .			
flags	. . .			
pkeyique_id	. . .			
pid_name	. . .			
. . .	. . .			
port_bw#RedWic40C12Bw	Number	16	yes	null
port_bw#ERedWic40C12Bw	Number	16	yes	null

810 { version ... pid_name }

806 { port_bw#RedWic40C12Bw port_bw#ERedWic40C12Bw }

FIG. 23

SUBSTITUTE SHEET (RULE 26)

columns:

name	type	length	null	index component	choices 123456789 123456
902 —— version	String	16	no	no	-
904 —— child_name	String	32	no	no	-
906 —— attribute_name	String	32	no	no	-
908 —— id_name	String	32	no	no	-
910 —— id_type	String	16	no	no	PRIMARY/SECONDARY
912 —— parent_hname	String	527	no	no	-
913 —— child_pkey	Number	16	-	no	-
915 —— child_id	Number	16	-	no	-
914 —— parent_hpkey	String	527	no	no	-
916 —— parent_hpid	String	527	no	no	-
917 —— expire_timestamp	Date	-	yes	no	-

COLUMN DEFINITIONS
FOR ud_contain

FIG. 24

columns:

name	type	length	null	index component	choices 123456789 123456
952 — version	String	16	no	no	-
954 — child_name	String	32	no	no	-
956 — attribute_name	String	32	no	no	-
958 — pool_name	String	32	no	no	-
959 — pool_type	String	16	no	no	ACTUAL/PSEUDO
960 — parent_hname	String	527	no	no	-
962 — parent_attr_name	String	32	no	no	-
963 — child_pkey	Number	16	-	no	-
965 — child_pool	Number	16	-	no	-
964 — parent_hpkey	String	527	no	no	-
966 — expire_timestamp	Date	-	yes	no	-

COLUMN DEFINITIONS
FOR ud_consume

FIG. 25

25/25

columns:

name	type	length	null	index component	choices 123456789 123456
1002——version	String	16	no	no	'
1004——src_name	String	32	no	no	'
1006——attribute_name	String	32	no	no	'
1008——ptr_name	String	32	no	no	'
1009——ptr_type	String	16	no	no	PERMANENT/TRANSIENT
1010——des_hname	String	527	no	no	'
1011——src_pkey	Number	16	'	no	'
1013——des_pkey	String	16	'	no	'
1012——des_hpkey	String	527	no	no	'
1014——des_order	Number	16	'	no	'
1015——expire_timestamp	Date	-	yes	no	'

COLUMN DEFINITIONS
FOR ud_connect

FIG. 26

INTERNATIONAL SEARCH REPORT	International application No.
	PCT/US00/12049

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) :G06F 12/00, 17/30, 17/50

US CL : 707/203; 717/11

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)

U.S. : 707/203; 717/11

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)

EAST

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 4,558,413 A (SCHMIDT et al) 10 December 1985.	NONE
A	US 5,740,405 A (DEGRAAF) 14 April 1998.	NONE
A	US 5,555,418 A (NILSSON et al) 10 September 1996.	NONE
A,P	US 6,003,039 A (BARRY et al) 14 December 1999.	NONE

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"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
"A"	document defining the general state of the art which is not considered to be of particular relevance		
"E"	earlier document published on or after the international filing date	"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
"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)	"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
"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	"&"	document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
05 JULY 2000	**19 SEP 2000**

Name and mailing address of the ISA/US	Authorized officer
Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231	UYEN LE
Facsimile No. (703) 305-3230	Telephone No. (703) 305-4134

Form PCT/ISA/210 (second sheet) (July 1998)★