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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
31 October 2002 (31.10.2002)

PCT

(10) International Publication Number
WO 02/086738 A1

(51) International Patent Classification[7]: G06F 15/00, 15/16, 17/30, H04H 1/00

(21) International Application Number: PCT/US02/12909

(22) International Filing Date: 25 April 2002 (25.04.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/286,466 25 April 2001 (25.04.2001) US
10/021,855 13 December 2001 (13.12.2001) US

(71) Applicant: **BEA SYSTEMS, INC.** [US/US]; 2315 North First Street, San Jose, CA 95131 (US).

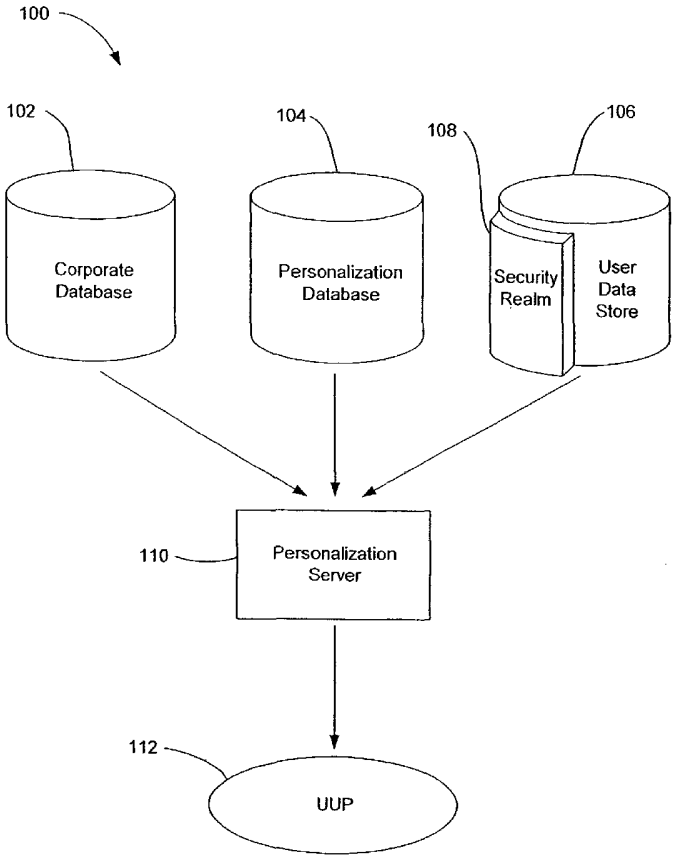
(72) Inventors: **BISSON, Michel**; 4905 Noble Park Place, Boulder, CO 80301 (US). **BREEDEN, Timothy**; 4945 West 128th Place, Broomfield, CO 80020 (US). **PACLAT, Charles**; 114 Wolcott Street, Medford, MA 02155 (US). **STAMM, Tom**; 894 West Willow Street, Louisville, CO 80027 (US). **WILLCOX, Steven**; 4604 Lee Hill Drive, Boulder, CO 80302 (US).

(74) Agents: **MEYER, Sheldon, R.** et al.; Fliesler Dubb Meyer and Lovejoy LLP, Suite 400, Four Embarcadero Center, San Francisco, CA 94111-4156 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZM, ZW.

[Continued on next page]

(54) Title: PERSONALIZATION SERVER UNIFIED USER PROFILE



(57) Abstract: The present invention includes systems utilizing, and methods for generating, a unified user profile (112) to provide a transparent interface to multiple data sources. A base user java bean is obtained to work through a personalization server (110) and access a personalization database (104). The base user java bean provides a transparent interface through which implicit and explicit properties can be retrieved and updated. An enterprise java bean is then created to extend the base user java bean such that the implicit and explicit properties can further be retrieved and updated from an external user database through the transparent interface.

WO 02/086738 A1

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

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.

PERSONALIZATION SERVER UNIFIED USER PROFILE

CLAIM OF PRIORITY

[0001] This application claims priority to U.S. Provisional patent application No. 60/286,466, filed April 25, 2001, entitled PERSONALIZATION SERVER UNIFIED USER PROFILE, incorporated herein by reference.

COPYRIGHT NOTICE

Field of the Invention:

[0003] The present invention relates generally to the integration of data from multiple sources.

Background of the Invention:

[0004] Corporations are continually looking for better ways to integrate new information with their existing data, such as information relating to current and prospective customers that may be acquired from third party sources. To make such integration effective, a company must be able to streamline the integration process, eliminate unnecessary cost or purchases, and eliminate the downtime that is typically necessary to implement a new data system or modify existing data structures.

[0005] As an example, a corporation may wish to incorporate additional user profile data into their established user data. This additional profile data may be configured and maintained by a separate source, such as a personalization server. The corporation often has pre-existing corporate customer or user data that is outside the scope of the personalization server. This data, which typically lives in an existing corporate database, may include, for each customer, information such as name, social security number, and/or company-particular information such as frequent flyer miles for a particular airline. The corporation would like to integrate this data seamlessly into their personalization solution, avoiding data migration difficulties where possible.

[0006] It is therefore an object of the invention to develop a seamless approach to the integration of data from an existing data source with data from an external source.

Summary of the Invention:

[0007] The present invention includes a system for generating a unified user profile. The system includes a first data source and a second data source. A server is used to access the first and second data sources. The server utilizes a user component adapted to aggregate data from the first and second data sources into a unified user profile.

[0008] Also included in the present invention is an architecture for generating a unified user profile. The architecture may be built on a base user enterprise java bean, which is capable of being extended to incorporate existing user data from a user data store. A user-specific enterprise java bean may then be generated, which allows transparent

read and write access to the existing user data.

[0009] The present invention also includes a method for generating a unified user profile. In one embodiment, a base user java bean is obtained that is adapted to work through a personalization server to access a personalization database. The base user java bean provides a transparent interface through which implicit and explicit properties can be retrieved and updated. An enterprise java bean is then created to extend the base user java bean such that the implicit and explicit properties can further be retrieved and updated from an external user database.

[0010] Also included in the present invention is a method for transparently accessing multiple data sources. In the method, a base user java bean is obtained that is adapted to work through a server to access an internal data source. The base user java bean provides a transparent interface through which implicit and explicit properties can be retrieved and updated to the internal data source. The user java bean is then extended such that said base user java bean further provides a transparent interface through which implicit and explicit properties can be retrieved and updated from at least one external data source.

[0011] The present invention further includes a system for transparently accessing multiple data sources. The system uses a server in communication with multiple data sources. An extended user java bean is included in the system that is adapted to provide transparent access to the data sources through the server.

Brief Description of the Figures:

[0012] **Figure 1** is an illustration of a UUP configuration in

accordance with one embodiment of the invention.

[0013] **Figure 2** is an illustration of a UUP configuration in accordance with one embodiment of the invention.

[0014] **Figure 3** is an illustration of a UUP configuration in accordance with one embodiment of the invention.

[0015] **Figure 4** is an illustration of a UUP configuration in accordance with one embodiment of the invention.

[0016] **Figures 5(a)** and **5(b)** are flowcharts showing steps for calling a setUserPoints() method for implicit and explict cases, in accordance with one embodiment of the present invention.

[0017] **Figure 6** is a flowchart showing steps for operating an ejbFind routine in accordance with one embodiment of the present invention.

Detailed Description of the Invention

[0018] In accordance with the foregoing summary of the invention, the following presents a detailed description of an embodiment of the present invention, which is presently considered to be the best mode.

[0019] An architecture of the present invention defines the way in which existing user data may be incorporated with more dynamically-changing personalization data. In a server, such as a personalization server that is used to personalize content or services for a particular use or group of users, system users are typically represented by user profiles. A user profile provides an ID for a user and access to the properties of a user, such as age or email address. Property values can be single-valued or multi-valued, and may be requested via a *getProperty()* function or

similar method which takes a property name as a key.

[0020] An advantage of a user profile of the present invention is that it may be extended and customized to retrieve user information from an existing data source. For example, a user profile that ships with a server or solution, such as a personalization server, may combine user properties, such as properties from a personalization server database with user properties from an LDAP server or legacy database as are known in the art, into a single user profile for use within an application. Developers and system users then need not worry about the different underlying data sources. The user profile is the only place necessary to go for user information.

[0021] A unified user profile (UUP) of the present invention includes this aggregation of properties from an existing data source and the personalization server database tables into a single, customized user profile. More specifically, a UUP marries existing user/customer data by extending a user component. By installing the personalization server database tables into the existing database instance and extending a user implementation, developers can quickly create a customized UUP that is capable of retrieving properties from, and storing/updating properties to, an existing database. This flexibility is desirable because it allows access to existing data without any migration of data or disruption of existing applications using that data. It should be understood, however, that existing data may be migrated into a separate personalization server database instance if desired.

[0022] One primary advantage of the UUP as compared to other server solutions is that the UUP requires no database scheme updates or data migration within a data management system, such as a customer

Relational Database Management System (RDBMS). The UUP is preferably created by writing an extension EJB, rather than by updating database tables, or running data migration scripts. Servers of the prior art often require the updating of the user database table schema for additional user properties.

[0023] **Figures 1-4** show possible configurations for a UUP system of the present invention. In a first configuration **100** of **Figure 1**, a corporate, legacy, or other external database **102** and personalization server database **104** provide property data to a personalization server **110**. The personalization server **110** also receives information from a user data store **106**, such as authentication information, user lists, group lists, and group membership. The user data store may be any appropriate system, such as an LDAP, Unix, or NT system as are known in the art. The user data store **106** also includes a security realm **108** for authentication. The personalization server database **104** and security realm **108** are kept separate in this configuration, as such separation of authentication and retrieval may be desirable, though not necessary to practice the invention. This configuration may be used where users and groups already exist in some type of data store, such as an LDAP directory. This existing user property data is then taken by the personalization server **110** and merged with the personalization data to generate the UUP **112**.

[0024] A second configuration **200**, as shown in **Figure 2**, may be useful where users and groups already exist in a user data store **204**, such as an LDAP directory, and no existing user data must be incorporated into the UUP **210**. All user and group property data is then preferably stored in the tables of the personalization server **202** database. The

personalization server **208** in this configuration still preferably utilizes a security realm **206** of the user data store **204**.

[0025] A third configuration **300**, shown in **Figure 3**, may be useful where there is no existing store of users and groups. The tables of the personalization server database **302** contain all user and group data, as well as preferably housing a separated security realm **304**. The personalization server **306** then only need to look to the personalization server database **302** in generating the UUP **308**.

[0026] A fourth configuration **400**, shown in **Figure 4**, may be useful where user, group, and property data are in an existing corporate, legacy, or other external database **402** and must be incorporated into the UUP **410** by the personalization server **408**. A custom security realm **404** must then be created in order to use the existing users and groups with the personalization server. The custom security realm need not necessarily be stored with the external database **402**, but may be incorporated into the personalization database **406**. Again, the retrieval and authentication realms are preferably kept separate.

[0027] One embodiment of an architecture of the present invention relies on three primary contributors for incorporating data in a UUP: (1) a base user enterprise java bean (EJB), (2) an user data store, and (3) a user-specific enterprise java bean (EJB).

[0028] A base user EJB is a Java class which is preferably extended by a personalization customer to incorporate existing user data into the personalization solution.

[0029] The base user EJB preferably provides a single, transparent interface through which both implicit properties and explicit properties can

be retrieved or updated. The base user EJB utilizes a property set, which may be used to give namespace qualifications to properties, as well as to define property types, allowable values, etc. A property set acts like a data schema for user properties. As used herein, transparency generally refers to the fact that a user or application can make a call or request without care as to where the data is stored or what naming convention the data may use. If the data is in a legacy database instead of a personalization database, the UUP will automatically process the request without the user or application ever needing to know about the location or name.

[0030] In one embodiment of the invention, subclasses of the base User EJB use two methods to retrieve or update: getProperty and setProperty. These methods may retrieve and/or update both implicit and explicit properties. The methods may be set as follows:

```
public Object getProperty(String propertySetName, String propertyName);
public void setProperty(String propertySetName, String propertyName,
        Object propertyValue);
```

These methods preferably use two primary attributes: propertySetName and propertyName. The propertySetName attribute specifies the data schema to which the invocation applies. In this way, the propertySetName acts as a namespace for the property that is to be retrieved or updated. The propertyName attribute specifies the name of the property to be updated or retrieved. One advantage of the using the propertySetName-propertyName pair is that a single propertyName may be used across multiple applications, or sub-application scopes. The multiple instances of

the property names may also correspond to differing definitions. Properties retrieved from the base User bean shall be referred to as *implicit properties*.

[0031] The user data store, which may be in existence prior to the incorporation, is typically a database or table where data is held that may relate to current users or customer data. This table may be colocated in a database instance of the personalization server. The existing user data store may hold user data which exists independent of the personalization server database tables. The personalization server may require the existing user data store to live in the same RDBMS instance as the personalization server tables.

[0032] An example "AcmeCustomer" RDBMS table is shown in Table 1. This table defines three values for each customer: (1) Customer_Name, (2) Acme_Points, and (3) Acme_Discount. Customer_Name is used as a unique identifier for each customer. This unique identifier, once integration is complete, is preferably used to uniquely identify a user throughout the Personalization server. Acme_Points is a sample customer value. An Acme customer might collect points as he or she makes purchases of Acme products. Acme_Discount is the discount the customer receives on each Acme product purchase.

Table 1 - AcmeCustomer RDBMS table

Customer_Name	Acme_Points	Acme_Discount
bsmith	50000	0.2
jpatadia	100000	0.1
ughandi	85000	0.15
mbisson	65000	0.3
kdickson	32000	0.05
tstamm	200000	0.2
tcook	100000	0.1

Once the existing data store is fully understood, an EJB that extends the base user EJB can be written, which takes advantage of the transparent value retrieval and update services. Methods of extending Java beans should be well known to persons skilled in the computer arts.

[0033] To integrate the existing user data with personalization tables provided with a personalization server, an EJB may be written to extend the user bean, which may be provided with the personalization server. Once this bean is completed, the personalization server client has transparent read and write access to properties previously stored in the user-specific database table (*explicit properties*), and to properties stored in the set of property tables of the Personalization server (*implicit properties*).

[0034] Continuing with the example "Acme Customer" data store, an AcmeUser EJB may be written that provides data update and retrieval mechanisms for the existing table. To operate within constraints of the

present invention, the AcmeUser EJB may define the following methods:

- public Long getAcmePoints() - Returns the number of Acme points collected to date by the customer.
- public void setAcmePoints(Long newAcmePointsValue) - Updates the number of Acme points for the customer.
- public Double getAcmeDiscount() - Returns the current discount for the customer.
- public void setAcmeDiscount() - Updates the customer's Acme discount.

Properties retrieved from the extended user bean are called *explicit properties.*

[0035] Once the methods of the extended bean are implemented, both the Acme points and the Acme discount may be retrieved with the inherited getProperty() and setProperty() methods implemented by the base User EJB.

[0036] Therefore, for example user *bsmith,* the following two methods would both return a Long containing the value 50000:

```
public Long getAcmePoints();
public Object getProperty (anyPropertySetName, "acmePoints");
```

Likewise, each of the following methods would update the Acme points

value for *bsmith* to 60,000:

```
public void setAcmePoints (new Double(60000));
public void setProperty (anyPropertySetName, "acmePoints", new
Double(60000));
```

[0037] In its implementation of transparent property update and retrieval, the UUP preferably uses the notion of Java reflection to determine whether a property is explicit before treating the property as *implicit* and employing the notion of Property Sets. Reflection is a feature of the Java programming language that allows an executing Java program to examine or introspect upon itself in order to manipulate internal properties of the program. An *explicit* property is preferably updated or retrieved before an implicit property, if the propertyName corresponds to an *explicit* property. Because of this search order, the actual property set name is of no consequence if an *explicit* property is being updated or retrieved.

[0038] The following example demonstrates a fictitious company's use of the UUP to take advantage of existing customer data. The example extends the User bean and retrieves data from a preexisting database. This example shows how, with relative ease, a customized UUP can be created that meets an application's persistence needs. The following table, Table 2, explains what may be extended in order to create a custom UUP.

Table 2 - Sample Extensions to create a custom UUP

Object	Must Extend
UUP Primary Key	UserPk--with no key fields added.
UUP EJB Interface	User
UUP EJB Implementation	UserImpl

[0039] The fact that a UUP is a ConfigurableEntity means that user profiles have the notion of setting and getting a property explicitly or implicitly. Explicitly setting a property as used herein means calling a setter method for a property directly. Implicitly setting a property means setting a property via the setProperty() method where no explicit setter method is available. For example, if a UUP contains a "userPoints" property, calling setUserPoints() directly would explicitly set the userPoints property. Calling setProperty() with the "userPoints" key would implicitly set the userPoints property. When called, setProperty() first looks for a setUserPoints() setter method to call in the user profile. If such a setter method exists, the method is called to set the property and do whatever is necessary regarding the change in value. Ultimately, it is the responsibility of the UUP implementation to persist explicitly-set property values, even if they are implicitly called via setProperty(). ConfigurableEntity preferably handles persisting implicitly set properties only where no explicit setter method exists.

[0040] **Figures 5(a)** and **5(b)** diagram both an explicit **550** and implicit **500** call to setUserPoints(), respectively. In both cases, it is the responsibility of the UUP bean to store the userPoints value. If no

setUserPoints() method had existed in the UUP bean, the ConfigurableEntity implementation would have handled storing the userPoints value. In the Implicit case of **Figure 5a**, a call is made to setProperty() **502**. The system checks to see if a setUserPoints() method exists **506**. If so, setUserPoints() is called **506**. If not, the system continues executing setProperty **510**. For the explicit case of **Figure 5b**, a call made to setUserPoints() **552** will simply call setUserPoints() **554**.

[0041] This notion of implicitly and explicitly setting properties allows for additional flexibility in UUP implementation. If any special logic needs to happen during the setting or getting of a property, such as the calculation of another value, it may be done using a setter or getter method for that property. Functionality external to the UUP may count on having a setProperty() method and getProperty() method for property access, eliminating any need to know whether a property has its own setter or getter. For example, a <um:getproperty> JSP tag may retrieve a userPoints property value even if a getUserPoints() method is the only way provided by the UUP to retrieve userPoints. This is because a getProperty() method of the UUP may first check to see if it has a getUserPoints() method before checking elsewhere. Properties that have an explicit set PropertyName() and get PropertyName() method are referred to as "explicit properties", while properties that can only be set through a call to setProperty() are referred to as "implicit properties".

[0042] When implementing a custom UUP EJB, it may only be necessary to implement explicit getter and setter methods for the explicit properties for the UUP. Implementations of these setters and getters would then set and retrieve the property values in the existing data store.

[0043] In one embodiment, a get PropertyName() / set PropertyName() approach is followed for all explicit property setting and getting in a UUP. If a UUP has an explicit userPoints property, an explicit getUserPoints() method is provided, as retrieveUserPoints() would not work. Similarly, setting userPoints is done with a setUserPoints() method. In this embodiment, the getProperty() and setProperty() methods look for getters and setters that follow this convention when getting and setting properties via implicit calls. Overriding setProperty() or getProperty() is not permitted. The getting and setting of explicit properties is done through getter and setter methods. Explicit getters and setters take and return objects. Primitives such as long and float are wrapped, such as in java.lang.Long and java.lang.Float objects, to be compatible with ConfigurableEntity's getProperty() and setProperty() methods.

[0044] If a getter method is provided, it may be a good idea to provide a setter method, and vice versa, as it cannot be predicted when a user or application will try to set or get a property. For example, if a getter is provided that retrieves a property from a database table without a corresponding setter, a call to setProperty() will store that property in a Personalization Server table. This is undesirable, as the value is retrieved from one place and set in another. The next time the property is retrieved, it would have its original value - not the value that was set. If a read-only property is to be provided, an empty setter method may be implemented.

[0045] A preferred definition of ConfigurableEntity's getProperty() method is as follows:

```
public Object getProperty(String propertySet,
                          String propertyName,
```

ConfigurableEntity *explicitSuccessor*,
Object *defaultValue*);

The getProperty() method preferably searches for properties in a specific order. For example, if a property is not found for a User, a Group may be queried for the value. In this case the User would inherit the property value from a Group. In ConfigurableEntity terms, the Group would be the User's "successor". If a property is not found in a ConfigurableEntity, then the successor to ConfigurableEntity may be queried. This way, ConfigurableEntities can inherit and override values from a parent entity.

[0046] Successors can be either implicit or explicit. An implicit successor is a default successor to a ConfigurableEntity, or a successor that is set for a specific Property Set. An explicit successor is preferably a ConfigurableEntity that may be passed as a parameter to the getProperty() method. Following is the order of the getProperty() property search as it exists in a preferred ConfigurableEntity:

- Look in the entity for the property for the specified Property Set.
- Look in the entity for the property in the default (null) Property Set.
- Look in the entity for the property in the Reserved Property Set (for properties from LDAP if using the LDAPRealm).
- Look for the property in the entity's explicit successor (if specified).
- Look for the property in the entity's successor for the specified Property Set.
- Look for the property in the entity's default successor.
- Look for a default value as defined in the Property Set if the Property Set is specified (not null).

- Return the defaultValue passed into the getProperty() method.

A preferred definition of ConfigurableEntity's setProperty() method is as follows:

```
public Object setProperty(String propertySet,
                          String propertyName,
                          Object value);
```

[0047] If, in this preferred method, setProperty() is used to set a property for a Property Set that is inconsistent with the Property Set definition, an exception may be thrown. For example, suppose a "UnifiedUserExample" Property Set is defined that has a userPoints property of type Integer. If someone tries to set the userPoints property for the "UnifiedUserExample" Property Set to be "abc" an exception would be thrown because userPoints is defined as being of type Integer and "abc" is text. Similarly, setting a Boolean property value to "bar" would result in an exception because Boolean values are restricted to Boolean objects.

[0048] If setProperty() is called and null is passed for the Property Set, the property value may be set in the null Property Set, referred to as the default Property Set. As described previously in the search order of getProperty(), the default property set is preferably searched before looking for the property value in the "Reserved" Property Set and successor.

[0049] The "Reserved" Property Set is preferably a read-only Property Set that may be used to hold property values from an external datastore. The "Reserved" Property Set may be used in the

Personalization Server, such as when properties are retrieved from an LDAP directory. Attempting to set a property in the "Reserved" Property Set may result in an exception being thrown. Properties in the "Reserved" Property Set and the Reserved Property Set itself may not be editable via User Management tools. Preferred User Management tools allow the specification of attributes to be retrieved from an LDAP or other server for users and groups. These attributes may then be the only ones retrieved at runtime.

[0050] Properties may be set via setProperty() with a Property Set specified that does not exist. This may be undesirable. When done, a Property Set is not created "on-the-fly" for the specified Property Set name. Rather, the specified Property Set name serves only as a namespace for the property. Similarly, it may be undesirable to set a property via setProperty() for an existing Property Set, specifying a property that does not exist for that Property Set. Properties set in either of these ways may not be editable through the User Management tools, while properties in the "null" or "default" property set may be editable.

[0051] A call to getProperty() preferably returns a java.lang.Long object if setProperty() is called passing a java.lang.Integer object value. Code retrieving such a property may be written as follows:

```
Object value = myUser.getProperty("my_property_set",
                                  "my_integer_property",
                                  null,
                                  null);
Number tempNumber = (Number) value;
int realValue =
```

```
tempNumber.intValue();
```

[0052] A call to getProperty() preferably returns a java.lang.Double object if setProperty() is called with a java.lang.Float object. Code retrieving such a property may be written as follows:

```
Object value     = myUser.getProperty("my_property_set",
                                      "my_float_property",
                                      null,
                                      null);
    Number tempNumber = (Number) value;
    float  realValue  = tempNumber.floatValue();
```

[0053] A User object preferably offers functionality for EJB find operations that makes integrating a UUP with the Personalization Server easy. **Figure 6** shows a flowchart for an ejbFind() operation. An extended UUP ejbFind() searches for records in the existing data store **602**. If successful, a call will be made to super.ejbFind() **604**, the User object ejbFind(). If successful, the User object ejbFind() will create the necessary records for the UUP in the Personalization Server database tables if they do not yet exist **610** and return the appropriate primary key. If the User object ejbFind() fails, it may check the underlying security realm **608** to determine whether the username corresponds to a valid user. If so, the User object ejbFind() creates the necessary records **610**, thereby eliminating finder errors and the time needed initially to migrate user data into the Personalization Server User database tables. If either ejbFind() fails or the user does not exist in the realm, a Finder Error is encountered

606. If no Finder Error is encountered, the appropriate primary key is returned **612**.

[0054] If the configuration is one such that the realm cannot verify the existence of the user, but the user must be created, it may be the responsibility of the EJB to create superclass records that are not found initially.

[0055] The last step in one embodiment of creating a custom UUP requires the UUP to be registered with a personalization or other server, such as through user management tools. In order to register the UUP, preferred user management tools utilize the following, in Table 3:

Table 3 - Registering the UUP (example)

Profile Type Name	**Arbitrary name that is later used to refer to the profile type through the User Management system's *<um:getprofile>* JSP extension tag**
Profile Home Class	The home class of the new profile type
Profile Remote Interface	The remote interface of the new profile type
Profile Primary Key Class	The primary key class of the new profile type
Profile JNDI Name	The JNDI lookup name of the new profile type

By registering the UUP with the Personalization Server, it becomes possible to ask for the new profile type with the <um:getprofile> JSP tag:

<um:getprofile profileType="UnifiedUserExample"

profileKey="<%=username%>"/>

[0056] It is then possible to use the <um:getproperty> and <um:setproperty> JSP tags with the UUP.

LDAP Property Retrieval Support

[0057] In addition to the transparent retrieval/update of implicit and explicit properties, one embodiment of a unified user profile mechanism facilitates the retrieval of user information from an LDAP server, with no Java code required from the personalization server customer. A set of administration tools allows specification of user and group properties to be retrieved from the LDAP server at a property request during application run time. Preferably, the only requirement of the Personalization Server customer for LDAP property retrieval is that the customer employ an LDAP security realm. At runtime, the UUP queries certain configuration information to detect whether the LDAP security realm is currently in use. If so, the names of the user and group properties to be retrieved may be obtained from an LDAPConfiguration session EJB, and the appropriate properties for the current user retrieved. The customer may not be required to use the LDAP security realm to receive the benefit of other UUP capabilities.

[0058] Other features, aspects and objects of the invention can be obtained from a review of the figures and the claims. It is to be understood that other embodiments of the invention can be developed and fall within the spirit and scope of the invention and claims.

[0059] The foregoing description of preferred embodiments of the

present invention has been provided for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Obviously, many modifications and variations will be apparent to the practitioner skilled in the art. The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, thereby enabling others skilled in the art to understand the invention for various embodiments and with various modifications that are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims and their equivalence.

CLAIMS

What is claimed is:

1. A system for generating a unified user profile to allow transparent access to multiple data sources, the system comprising:

(a) a first data source;

(b) a second data source; and

(c) a server adapted to access said first and second data source, said server comprising a component adapted to aggregate data from said first and second data sources into a unified user profile.

2. A system according to claim 1, wherein said first data source is selected from the group consisting of legacy databases, corporate databases, and user data stores.

3. A system according to claim 1, wherein said first data source contains data selected from the group consisting of authentication information, user lists, group lists, and group membership.

4. A system according to claim 1, further comprising a security realm adapted to allow authentication of data in at least one of said first and second data sources.

5. A system according to claim 1, wherein said server is a personalization server.

6. A system according to claim 1, wherein said component comprises an enterprise java bean.

7. A system according to claim 6, wherein said enterprise java bean retrieves and updates data in at least one of said first and second data sources using methods selected from the group consisting of getProperty() and setProperty().

8. A system according to claim 1, wherein said component comprises an extended java bean.

9. A system according to claim 1, wherein said component provides a transparent interface through which implicit and explicit properties can be retrieved and updated.

10. A system according to claim 9, wherein said component comprises a property set, said property set adapted to give namespace qualifications to said implicit and explicit properties.

11. A system according to claim 1, wherein said component comprises getter and setter properties.

12. A system according to claim 1, wherein said component provides a transparent interface adapted to store and retrieve data from said first data store and said second data store.

13. A system according to claim 1, wherein said second data source is a personalization database.

14. An architecture for generating a unified user profile for transparent access to existing user data, the architecture comprising:

(a) a base user enterprise Java bean, said base user enterprise Java bean capable of being extended to incorporate said existing user data;

(b) a user data store adapted to contain said existing user data; and

(c) a user-specific enterprise java bean, adapted to provide transparent read and write access to said existing user data.

15. An architecture according to claim 14, further comprising a data source containing data external to said existing user data.

16. An architecture according to claim 15, wherein said user-specific enterprise Java bean further allows transparent read and write access to said data in said data source.

17. An architecture according to claim 14, further comprising a server adapted to provide said read and write access to a user of said unified user profile.

18. An architecture according to claim 17, wherein said server is a personalization server.

19. An architecture according to claim 14, wherein said user data store is a table in an internal data source selected from the group consisting of legacy databases, corporate databases, and customer databases.

20. An architecture according to claim 14, wherein said user data store contains data selected from the group consisting of authentication information, user lists, group lists, and group membership.

21. An architecture according to claim 14, further comprising a security realm adapted to allow authentication of data in said user data store.

22. An architecture according to claim 14, wherein said user-specific enterprise Java bean utilizes a property set, said property set adapted to give namespace qualifications to implicit and explicit properties of said existing user data.

23. An architecture according to claim 14, wherein said user-specific

enterprise Java bean utilizes getter and setter properties.

24. A method for generating a unified user profile for providing transparent access to a personalization database and external user database, said method comprising the steps of:

(a) obtaining a base user java bean adapted to work through a personalization server to access said personalization database, said base user java bean adapted to provide a transparent interface through which implicit and explicit properties can be retrieved and updated from the personalization database; and

(b) creating an enterprise java bean to extend the base user java bean such that said implicit and explicit properties can further be retrieved and updated from an external user database.

25. A method according to claim 24, further comprising the step of generating transparent read and write access to said external database through the extended said base user java bean.

26. A method according to claim 24, further comprising the step of configuring a server to provide said read and write access.

27. A method according to claim 26, wherein said server is a personalization server.

28. A method according to claim 24, wherein said external user database is selected from the group consisting of legacy databases, corporate databases, and customer databases.

29. A method according to claim 24, wherein said external user database contains data selected from the group consisting of authentication information, user lists, group lists, and group membership.

30. A method according to claim 24, further comprising the step of obtaining a security realm adapted to allow authentication of data in said personalization database and said external user database.

31. A method according to claim 24, wherein the extended base user java bean utilizes a property set, said property set adapted to give namespace qualifications to implicit and explicit properties of said data in said personalization database.

32. A method according to claim 31, wherein said implicit and explicit properties comprise getter and setter properties.

33. A method for transparently accessing multiple data sources, said method comprising the steps of:

(a) obtaining a base user java bean adapted to work through a

server to access an internal data source, said base user java bean adapted to provide a transparent interface through which implicit and explicit properties can be retrieved and updated; and

(b) extending the user java bean such that said base user java bean is further adapted to provide a transparent interface through which implicit and explicit properties can be retrieved and updated from at least one external data source.

34. A method according to claim 33, further comprising the step of configuring a server to operate said transparent interface.

35. A method according to claim 33, further comprising the step of obtaining a security realm adapted to allow authentication of data in said internal data source and said external data source.

36. An method according to claim 33, further comprising the step of configuring a property set for the extended user java bean.

37. A method according to claim 35, wherein said property set is adapted to give namespace qualifications to implicit and explicit properties of said data in said internal and external data sources.

38. A method according to claim 37, wherein said implicit and explicit

properties comprise getter and setter properties.

39. A method according to claim 37, further comprising the step of using reflection to determine whether a property of said data in said internal and external data sources is explicit.

40. A system for transparently accessing multiple data sources, said system comprising:

(a) a plurality of data sources;

(b) a server in communication with each said data source; and

(c) an extended user java bean adapted to provide transparent access to said plurality of data sources through said server.

41. A system according to claim 40, wherein at least one of said plurality of data sources is selected from the group consisting of legacy databases, corporate databases, and user data stores.

42. A system according to claim 40, further comprising a security realm adapted to allow authentication of data in at least one of said plurality of data sources.

43. A system according to claim 40, wherein said server is a personalization server.

44. A system according to claim 40, wherein said extended user java bean retrieves and updates data in at least one of said plurality of data sources using methods selected from the group consisting of getProperty() and setProperty().

45. A system according to claim 40, wherein said extended user java bean is adapted to allow implicit and explicit properties of data in said plurality of data sources to be retrieved and updated.

46. A system according to claim 45, wherein said extended user java bean utilizes a property set, said property set adapted to give namespace qualifications to said implicit and explicit properties.

47. A system according to claim 45, wherein said implicit and explicit properties comprise getter and setter properties.

48. A system for unifying multiple data sources, said system comprising:

(a) a naming convention to be followed in storing and accessing data in the data sources;

(b) a plurality of data sources, at least one data source containing a data entry not following said naming convention;

(c) a set of identifier pairs, each identifier pair corresponding to a data entry that does not follow said naming convention, the identifier pair including the name of the entry and a

corresponding name that follows the naming convention; and

(d) a server in communication with each data source and the set of identifier pairs, the server adapted to allow access to the data sources by a request following said naming convention.

49. A system according to claim 48, wherein at least one of said plurality of data sources is selected from the group consisting of legacy databases, corporate databases, and user data stores.

50. A system according to claim 48, further comprising a security realm adapted to allow authentication of data in at least one of said plurality of data sources.

51. A system for generating a unified user profile adapted to allow transparent access to multiple data sources, the system comprising a server including:

(a) a first component adapted to access a first data source;

(b) a second component adapted to access a second data source; and

(c) a user component adapted to aggregate data from the first and second data sources into a unified user profile.

52. A system according to claim 51, further comprising component adapted to access a security realm for authentication of data in at least one

of said first and second data sources.

53. A system according to claim 51, wherein the user component comprises an enterprise java bean.

54. A system according to claim 51, wherein the user component retrieves and updates data in at least one of the first and second data sources using methods selected from the group consisting of getProperty() and setProperty().

55. A system according to claim 51, wherein the user component provides a transparent interface through which implicit and explicit properties can be retrieved and updated.

56. A system according to claim 55, wherein the user component comprises a property set, said property set adapted to give namespace qualifications to said implicit and explicit properties.

57. A system according to claim 51, wherein the user component comprises getter and setter properties.

58. An architecture for generating a profile adapted to provide access to user data, the architecture comprising:

(a) a base user enterprise Java bean, said base user enterprise

Java bean capable of incorporating the user data; and

(b) a user-specific enterprise java bean, adapted to provide transparent read and write access to the user data.

59. An architecture according to claim 58, further comprising a server adapted to provide the read and write access to the user data.

60. An architecture according to claim 58, wherein said user data store contains data selected from the group consisting of authentication information, user lists, group lists, and group membership.

61. An architecture according to claim 58, wherein said user-specific enterprise Java bean utilizes a property set, said property set adapted to give namespace qualifications to implicit and explicit properties of the user data.

62. An architecture according to claim 59, wherein the user-specific enterprise Java bean utilizes getter and setter properties.

63. A computer readable medium containing instructions which, when executed by a server, cause the server to perform the steps of:

(a) obtaining a base user java bean adapted to work through the server to access a first database, said base user java bean adapted to provide a transparent interface through which

implicit and explicit properties can be retrieved and updated from the first database; and

(b) creating an enterprise java bean to extend the base user java bean such that said implicit and explicit properties can further be retrieved and updated from a second database.

64. A computer readable medium according to claim 63, wherein the medium further causes the server to generate transparent read and write access to the second database through the extended said base user java bean.

65. A computer readable medium according to claim 63, wherein the medium further causes the server to obtain a security realm adapted to allow authentication of data in the first database and the second database.

66. A computer readable medium according to claim 63, wherein the extended base user java bean utilizes a property set, said property set adapted to give namespace qualifications to implicit and explicit properties of said data in the first database.

67. A computer readable medium according to claim 63, wherein the extended base user java bean utilizes getter and setter properties.

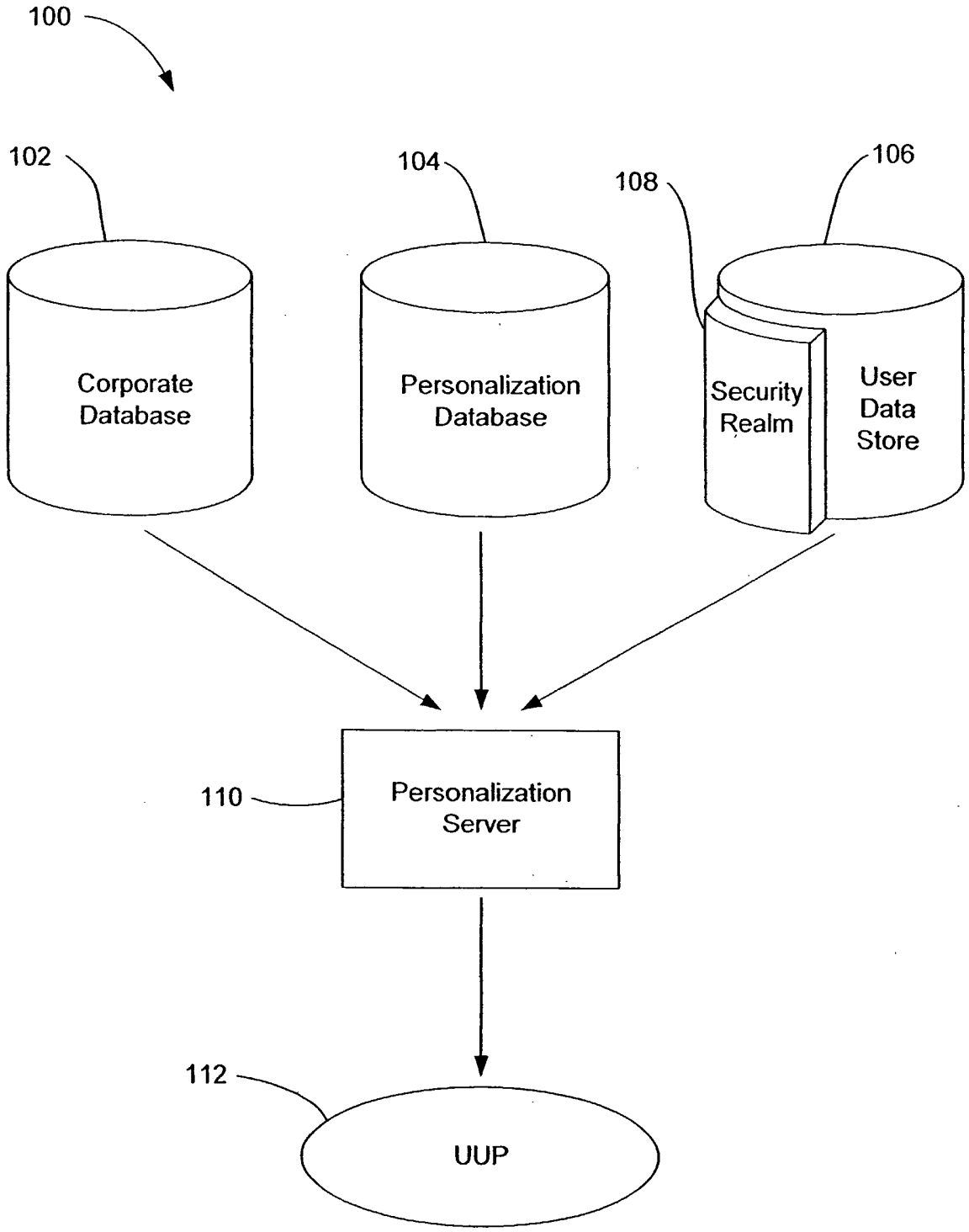

100
102
104
106
108
Corporate Database
Personalization Database
Security Realm
User Data Store
110
Personalization Server
112
UUP

FIG. - 1

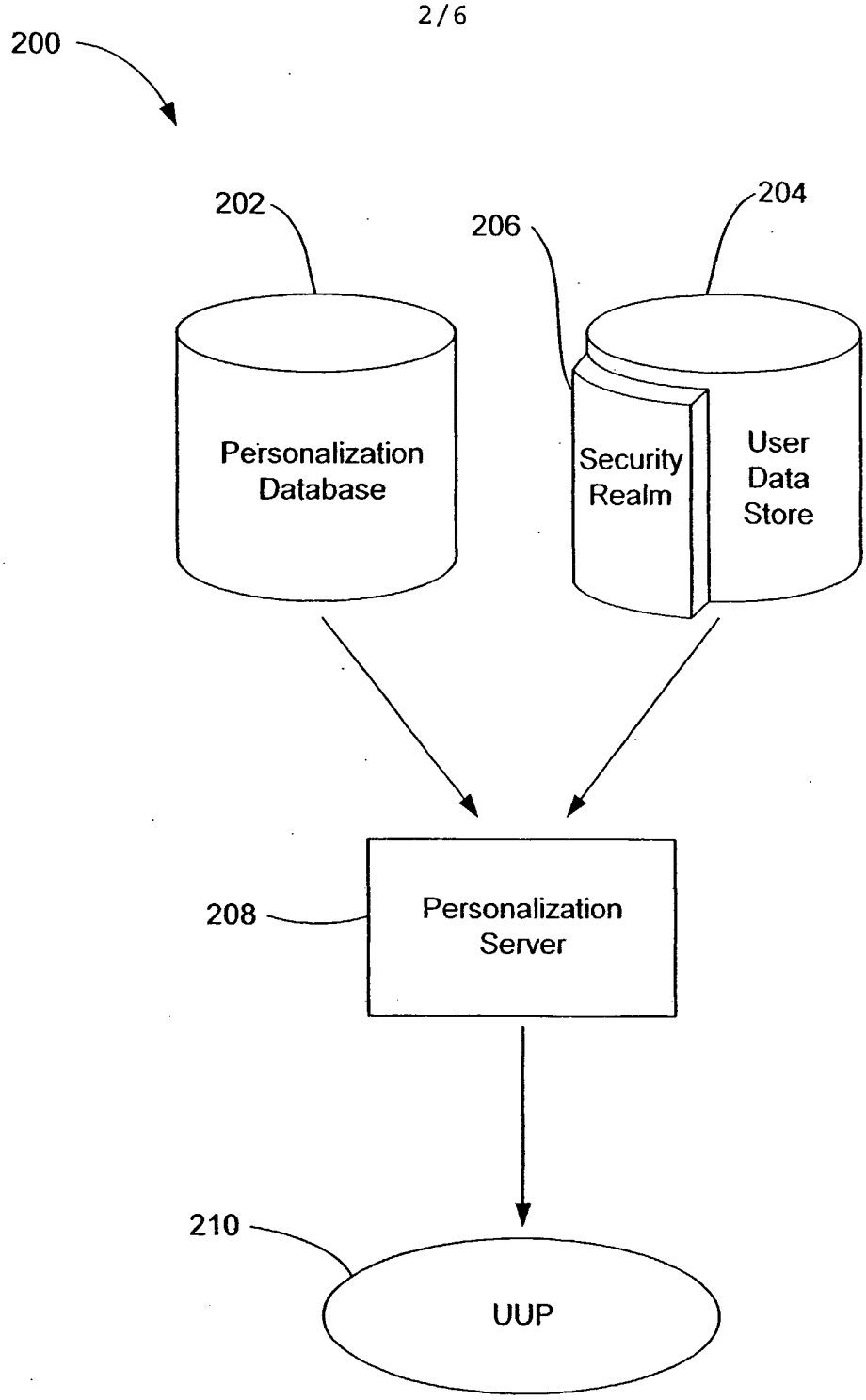

200
202
204
206
Personalization Database
Security Realm
User Data Store
208
Personalization Server
210
UUP

FIG. - 2

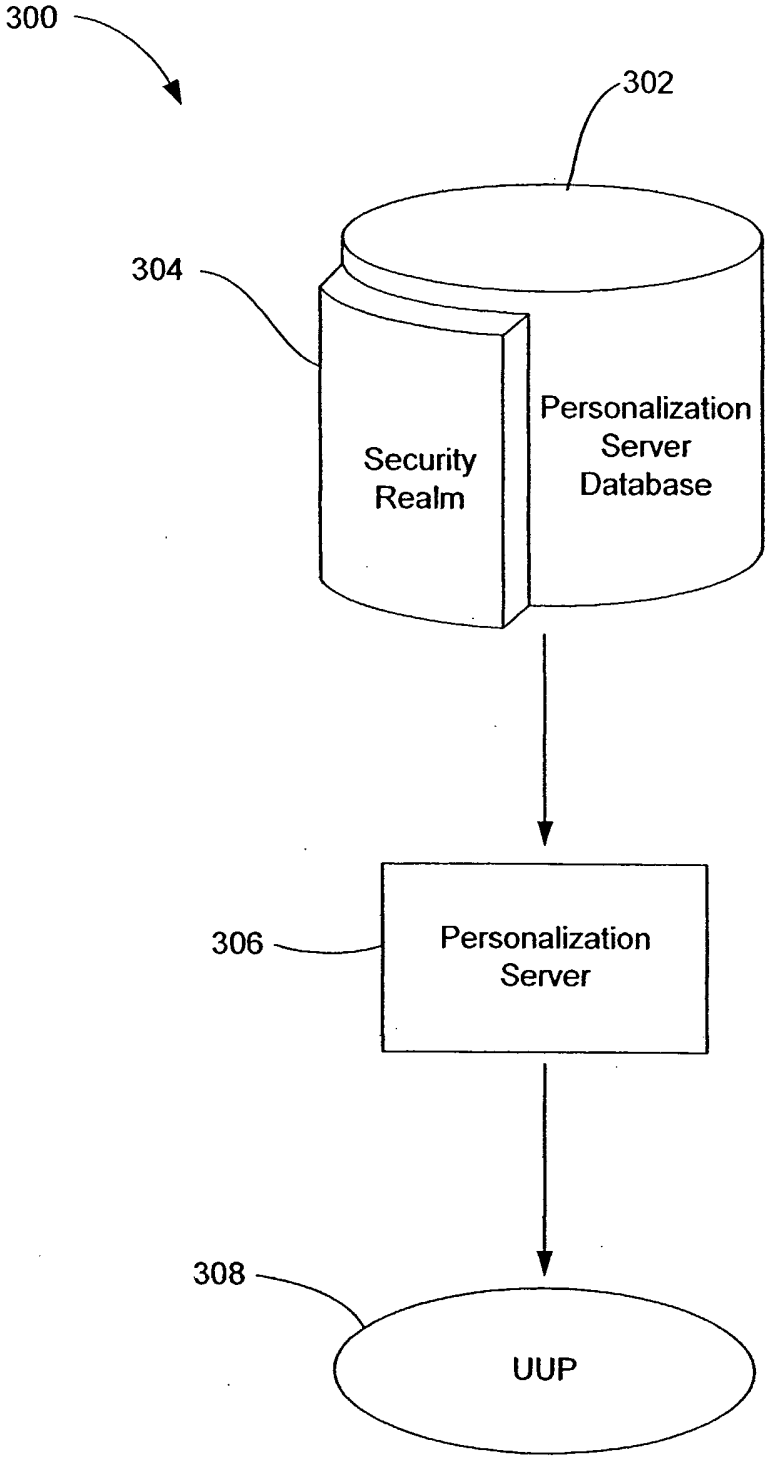

300
302
304
Security Realm
Personalization Server Database
306
Personalization Server
308
UUP

FIG. - 3

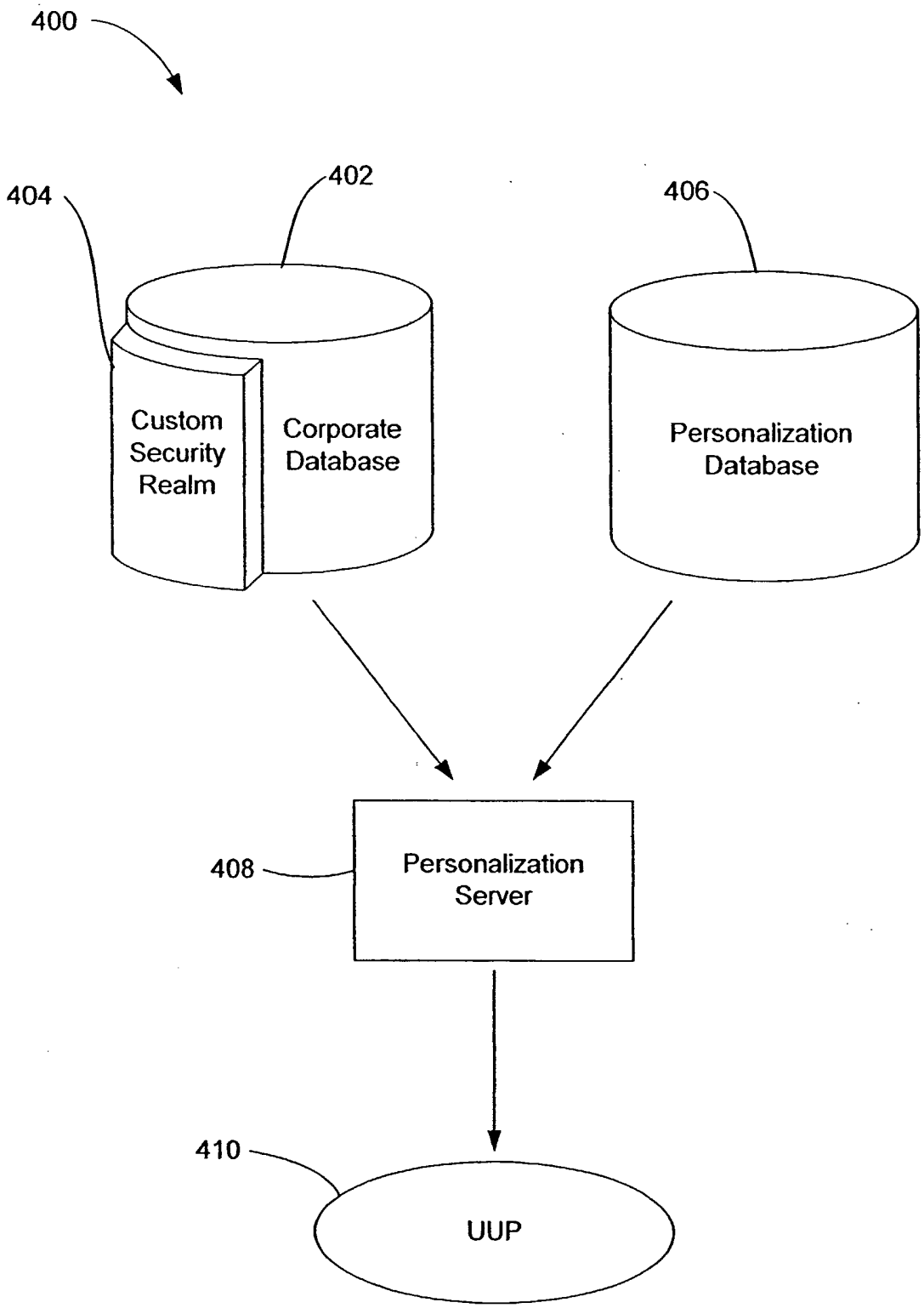

400
404
402
406
Custom Security Realm
Corporate Database
Personalization Database
408
Personalization Server
410
UUP

FIG. - 4

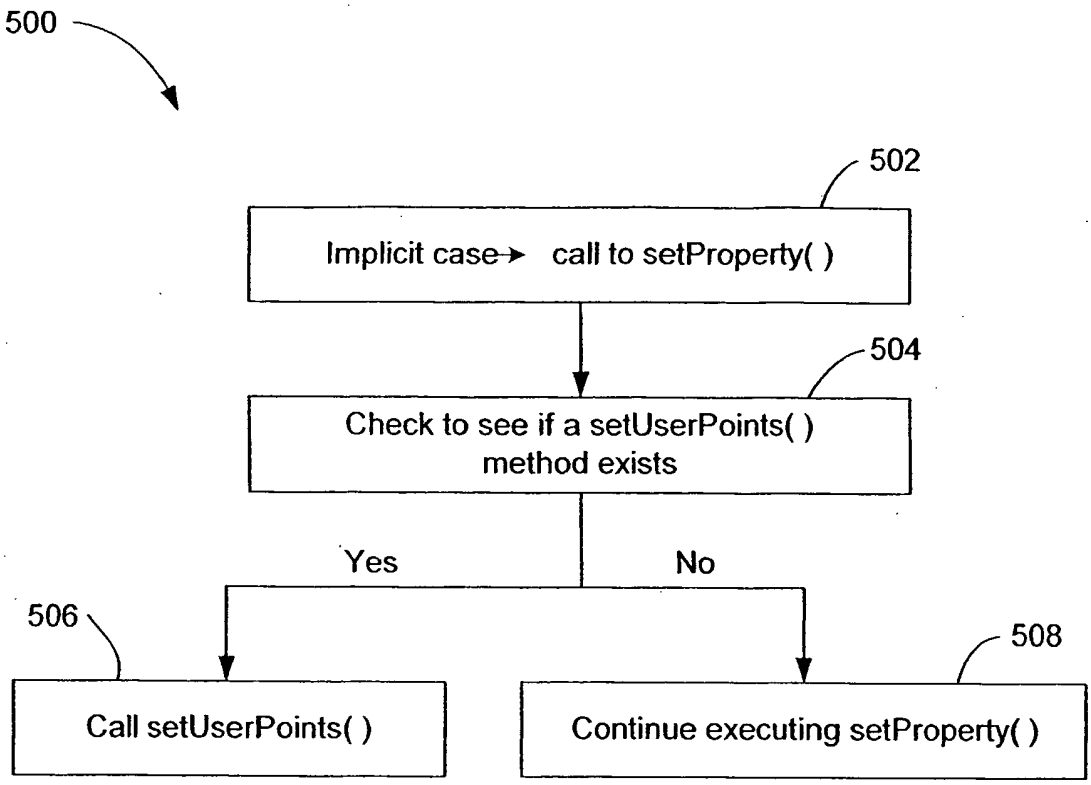

500
502
Implicit case→ call to setProperty( )
504
Check to see if a setUserPoints( ) method exists
Yes
No
506
508
Call setUserPoints( )
Continue executing setProperty( )

FIG. - 5a

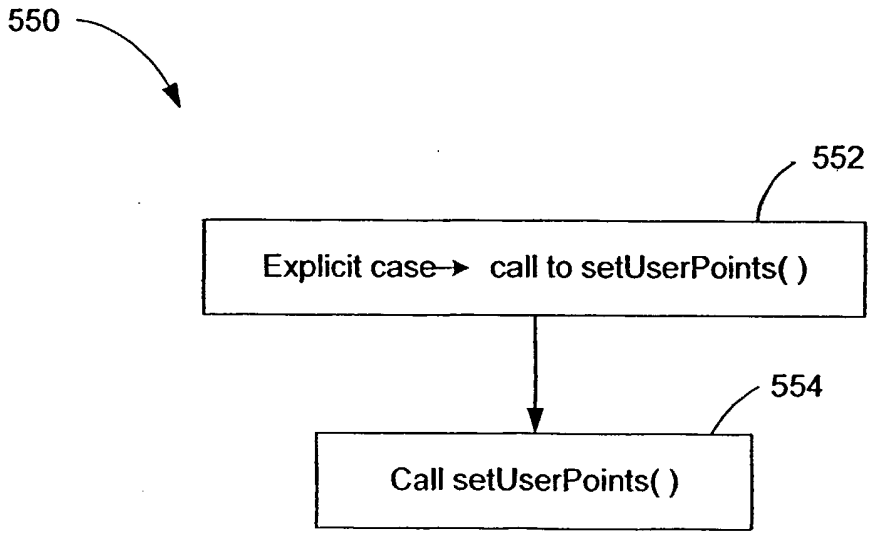

550
552
Explicit case→ call to setUserPoints( )
554
Call setUserPoints( )

FIG. - 5b

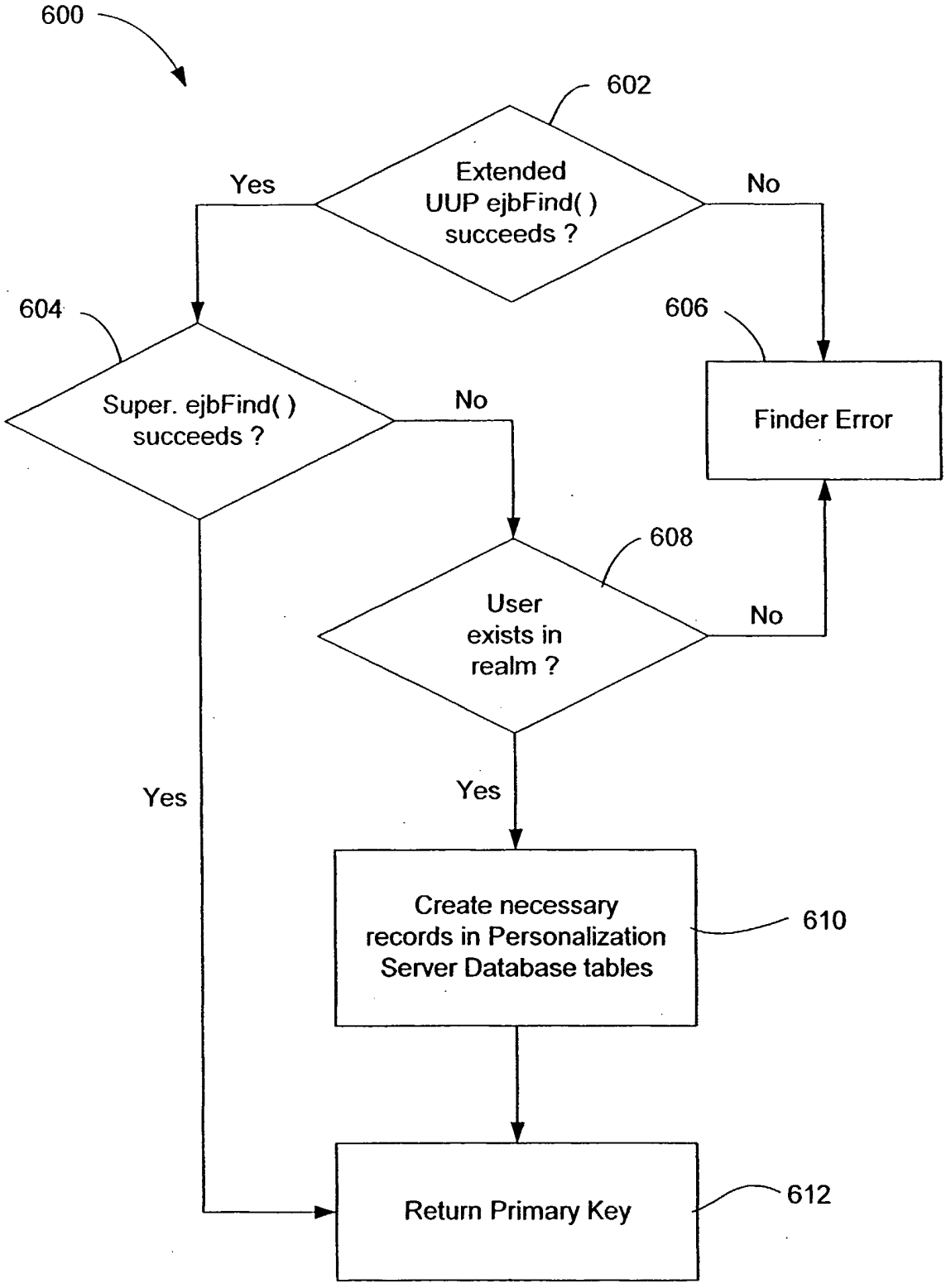

600
602
Extended UUP ejbFind( ) succeeds ?
Yes
No
604
Super. ejbFind( ) succeeds ?
606
Finder Error
No
608
User exists in realm ?
No
Yes
Yes
Create necessary records in Personalization Server Database tables
610
Return Primary Key
612

FIG. - 6

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US02/12909

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) :G06F 15/00, 15/16, 17/30; H04H 1/00
US CL :707/2, 10; 455/3.04; 709/203, 228
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. : 707/2, 10; 455/3.04; 709/203, 228

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

NONE

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

WEST

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 5,812,865 A (THEIMER et al.) 22 September 1998, the entire paper is relevant	1-67
Y	US 6,199,099 B1 (GERSHMAN et al.) 06 March 2001, the entire paper is relevant	1-67
Y	US 5,754,939 A (HERZ et al.) 19 May 1998, the entire paper is relevant	1-67
Y	US 6,195,651 B1 (HANDEL et al.) 27 February 2001, the entire paper is relevant	1-67
Y	US 5,813,006 A (POLNEROW et al.) 22 September 1998, the entire paper is relevant	1-67

☐ 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	"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
"E"	earlier document published on or after the international filing date	"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
"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)	"&"	document member of the same patent family
"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		

Date of the actual completion of the international search 14 JUNE 2002	Date of mailing of the international search report 11 JUL 2002
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer FOR THUY PARDO Telephone No. (703) 305-1091

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