US 20090037337A1

(19) **United States**

(12) **Patent Application Publication**

Baitalmal et al.

(10) **Pub. No.: US 2009/0037337 A1**

(43) **Pub. Date: Feb. 5, 2009**

(54) **SOFTWARE LICENSING AND ENFORCEMENT SYSTEM**

(76) Inventors: **Ahmad Baitalmal**, Renton, WA (US); **Daniel J. Kolke**, North Bend, WA (US); **Jon K. Collette**, Normandy Park, WA (US); **Casey Tompkins**, Woodinville, WA (US)

Correspondence Address:
MORGAN, LEWIS & BOCKIUS, LLP.
2 PALO ALTO SQUARE, 3000 EL CAMINO REAL
PALO ALTO, CA 94306 (US)

(21) Appl. No.: **12/182,800**

(22) Filed: **Jul. 30, 2008**

Related U.S. Application Data

(63) Continuation-in-part of application No. 12/102,854, filed on Apr. 14, 2008.

(60) Provisional application No. 60/962,877, filed on Jul. 31, 2007.

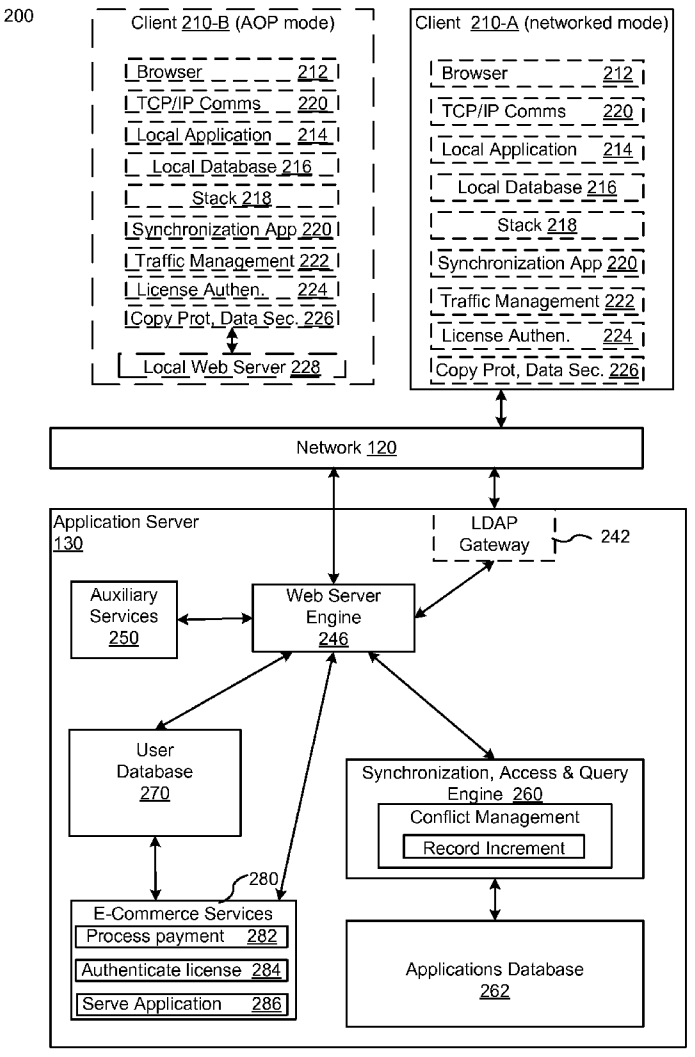
Publication Classification

(51) **Int. Cl.**
G06Q 99/00 (2006.01)

(52) **U.S. Cl.** **705/59**

(57) **ABSTRACT**

In an embodiment, a computer implemented method is described. The method is performed at one or more servers, hosting a marketplace application. A software application is received from a vendor for distribution. License terms are generated in response to a selection by the vendor from options provided by the marketplace application. The license terms are associated with the software application. The software application is made available for distribution through the marketplace application, in accordance with the license terms.



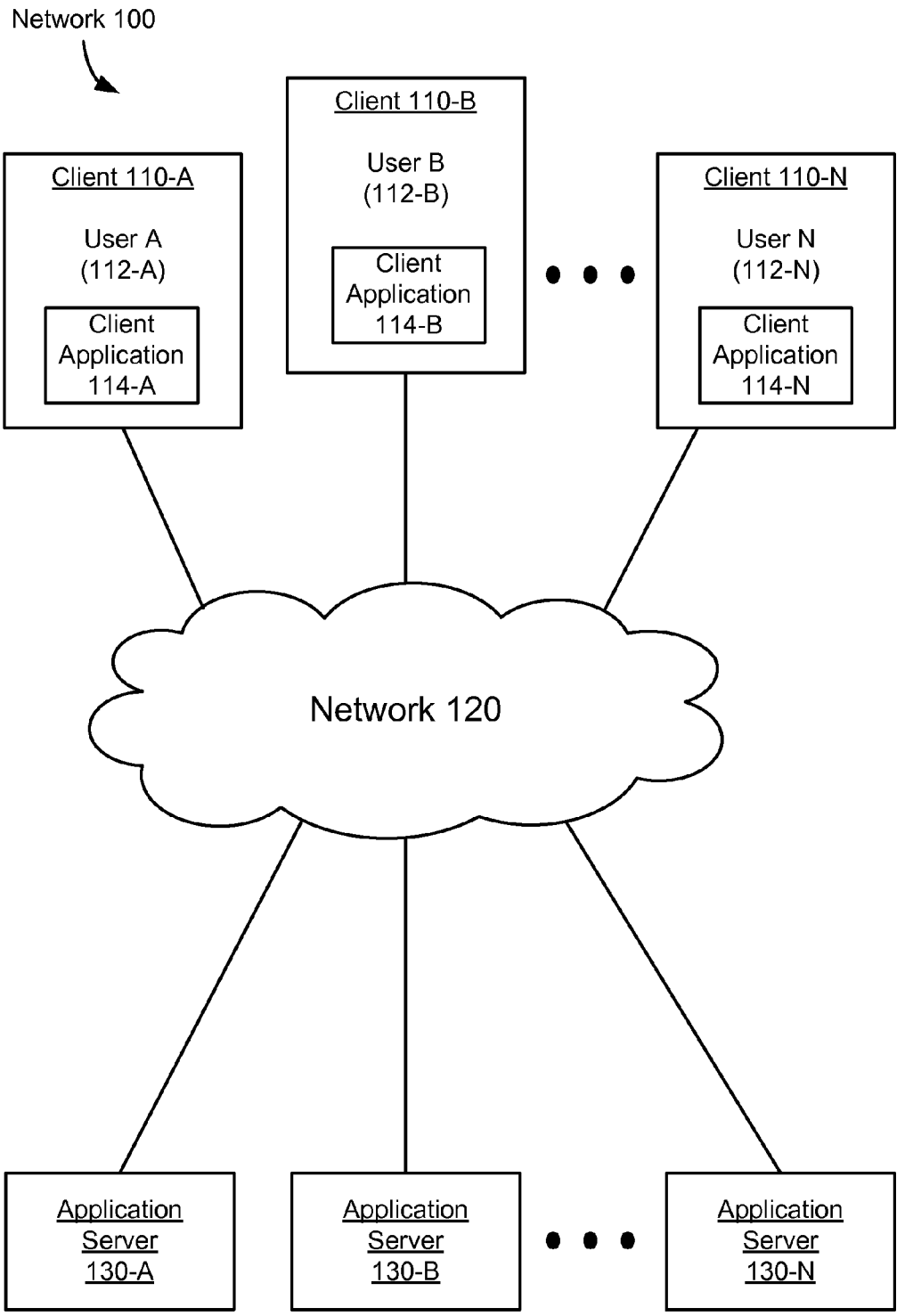

Network 100
Client 110-A
User A
(112-A)
Client Application 114-A
Client 110-B
User B
(112-B)
Client Application 114-B
Client 110-N
User N
(112-N)
Client Application 114-N
Network 120
Application Server 130-A
Application Server 130-B
Application Server 130-N

Figure 1

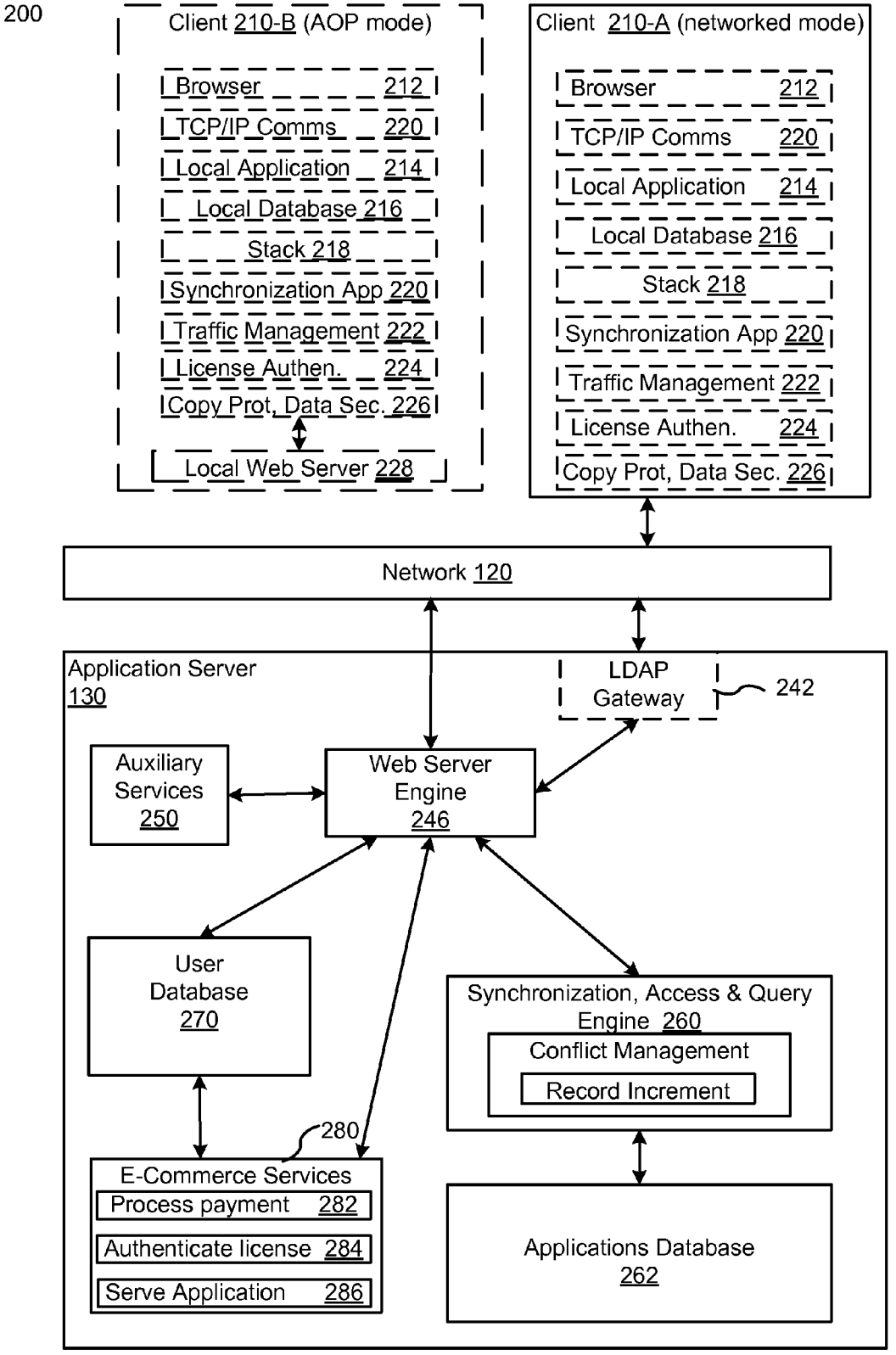

200
Client 210-B (AOP mode)
Browser 212
TCP/IP Comms 220
Local Application 214
Local Database 216
Stack 218
Synchronization App 220
Traffic Management 222
License Authen. 224
Copy Prot, Data Sec. 226
Local Web Server 228
Client 210-A (networked mode)
Browser 212
TCP/IP Comms 220
Local Application 214
Local Database 216
Stack 218
Synchronization App 220
Traffic Management 222
License Authen. 224
Copy Prot, Data Sec. 226
Network 120
Application Server 130
LDAP Gateway
242
Auxiliary Services 250
Web Server Engine 246
User Database 270
Synchronization, Access & Query Engine 260
Conflict Management
Record Increment
280
E-Commerce Services
Process payment 282
Authenticate license 284
Serve Application 286
Applications Database 262

Figure 2

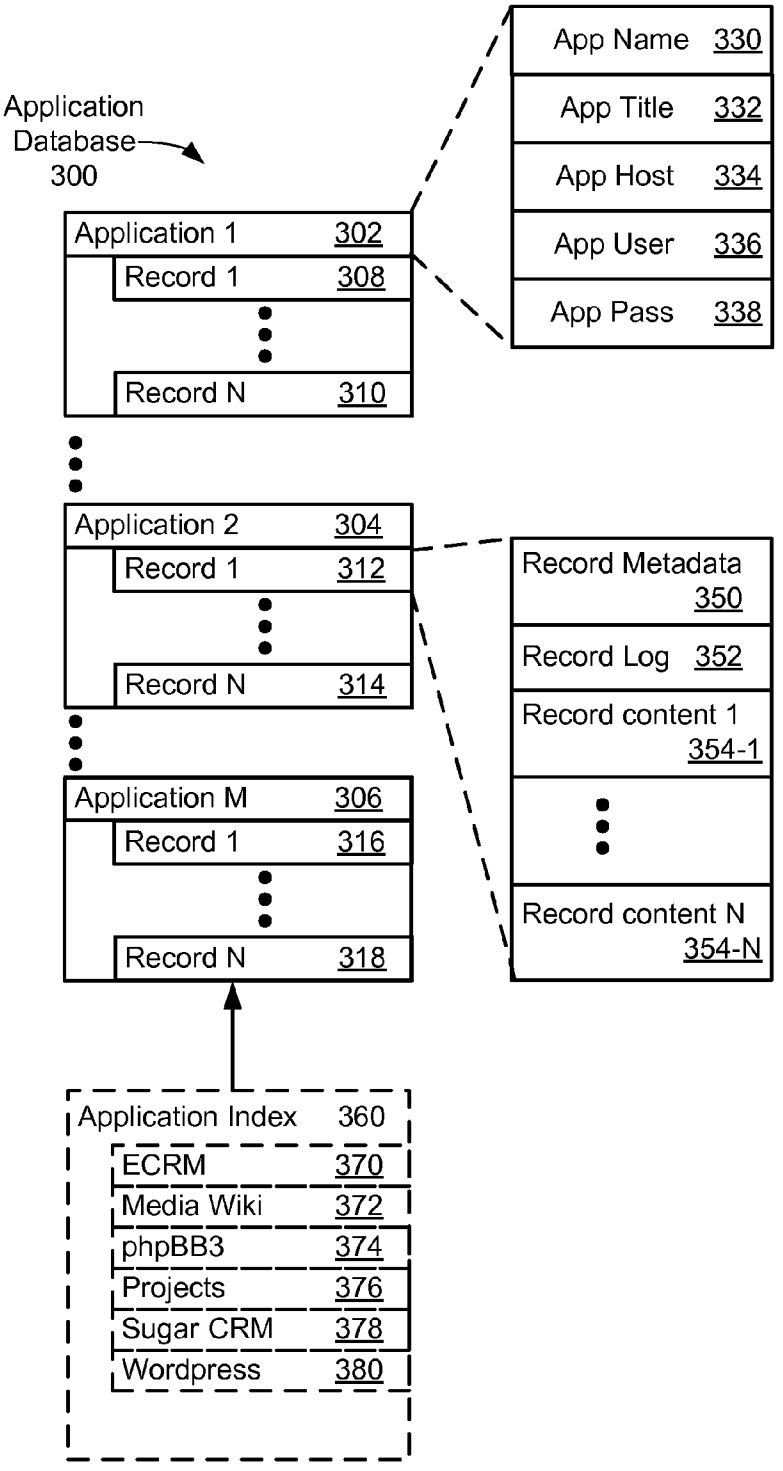

Application Database 300
Application 1 302
Record 1 308
Record N 310
Application 2 304
Record 1 312
Record N 314
Application M 306
Record 1 316
Record N 318
App Name 330
App Title 332
App Host 334
App User 336
App Pass 338
Record Metadata 350
Record Log 352
Record content 1 354-1
Record content N 354-N
Application Index 360
ECRM 370
Media Wiki 372
phpBB3 374
Projects 376
Sugar CRM 378
Wordpress 380

Figure 3

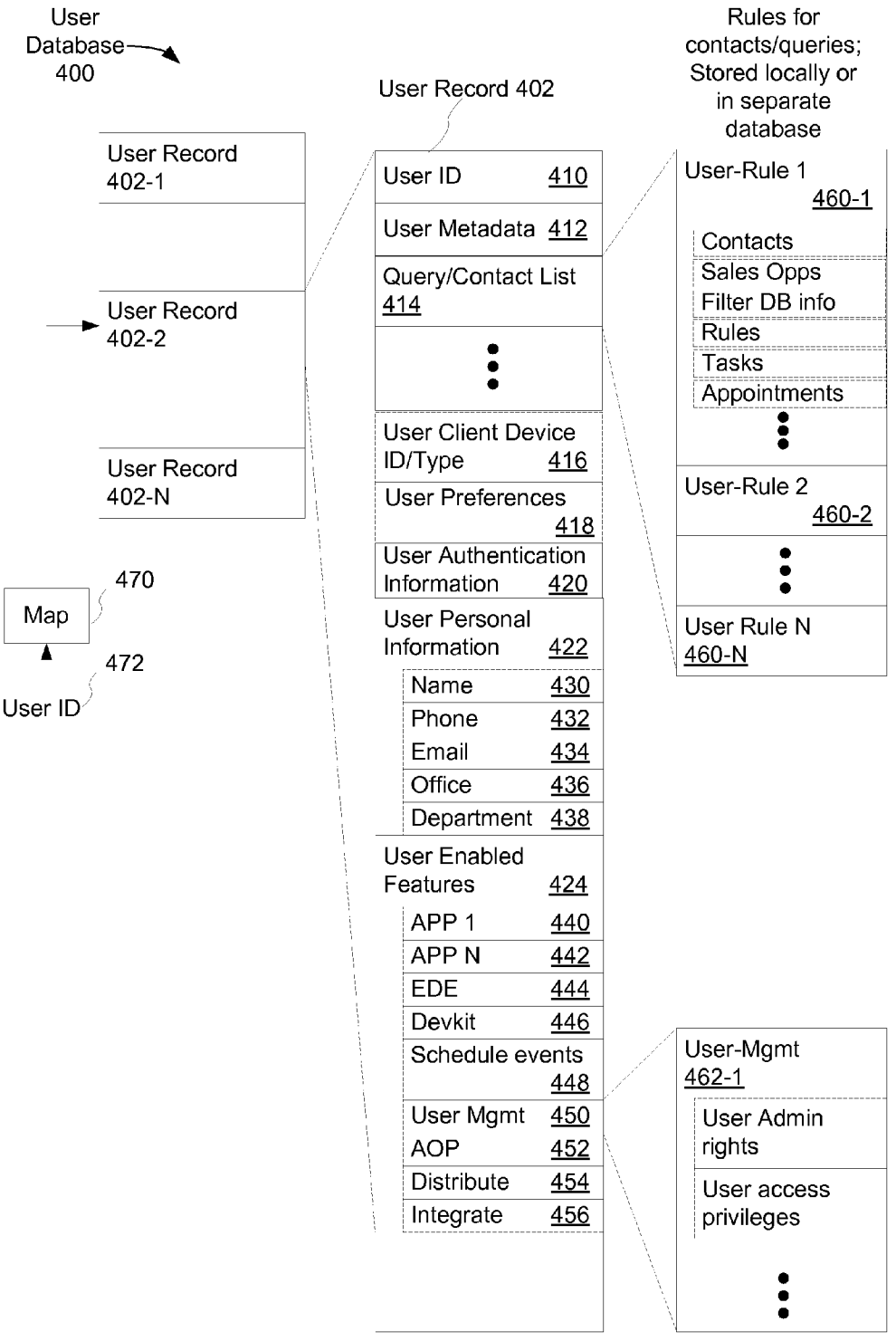

User Database 400
User Record 402-1
User Record 402-2
User Record 402-N
Map
470
472
User ID
User Record 402
User ID 410
User Metadata 412
Query/Contact List 414
User Client Device ID/Type 416
User Preferences 418
User Authentication Information 420
User Personal Information 422
Name 430
Phone 432
Email 434
Office 436
Department 438
User Enabled Features 424
APP 1 440
APP N 442
EDE 444
Devkit 446
Schedule events 448
User Mgmt 450
AOP 452
Distribute 454
Integrate 456
Rules for contacts/queries; Stored locally or in separate database
User-Rule 1 460-1
Contacts
Sales Opps Filter DB info
Rules
Tasks
Appointments
User-Rule 2 460-2
User Rule N 460-N
User-Mgmt 462-1
User Admin rights
User access privileges

Figure 4

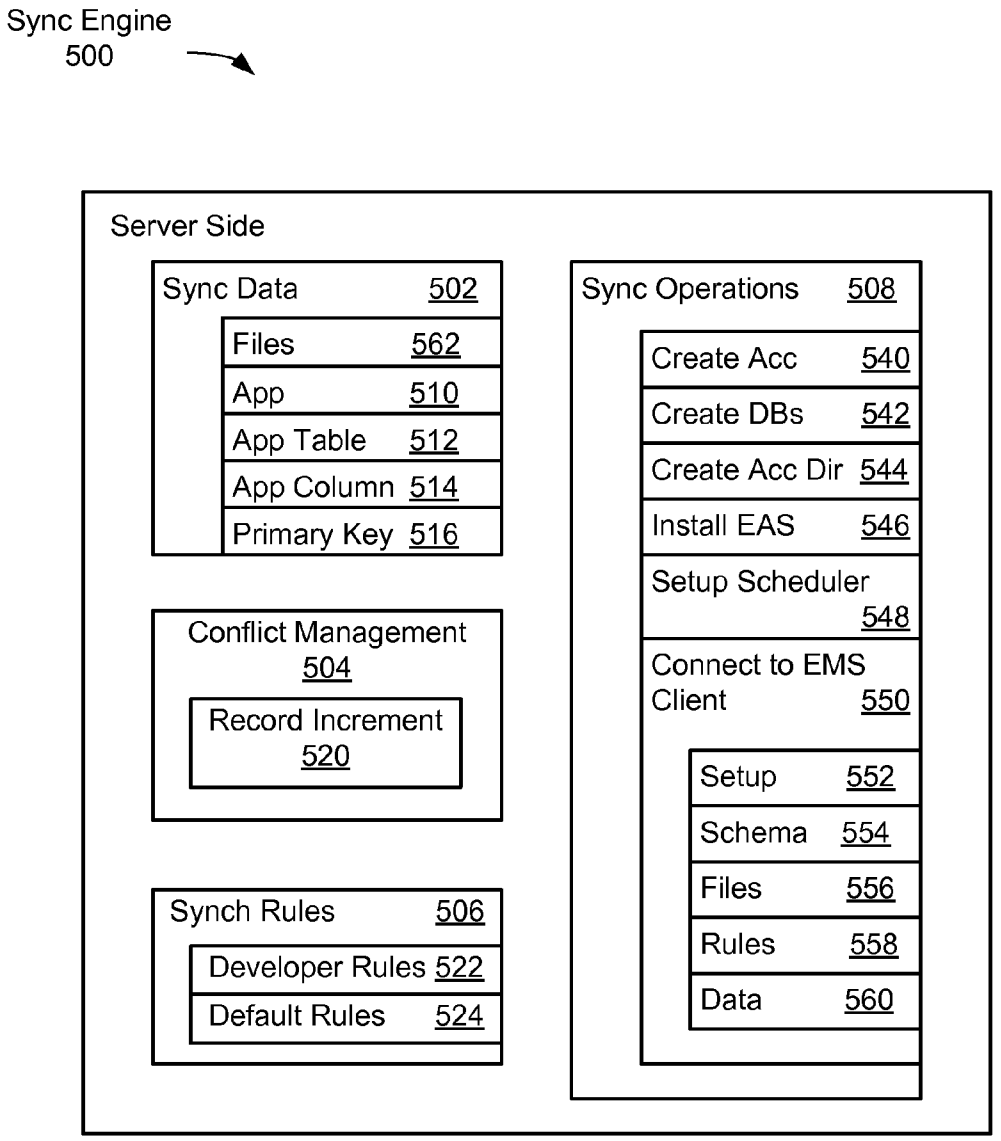

Sync Engine
500
Server Side
Sync Data 502
Files 562
App 510
App Table 512
App Column 514
Primary Key 516
Conflict Management
504
Record Increment
520
Synch Rules 506
Developer Rules 522
Default Rules 524
Sync Operations 508
Create Acc 540
Create DBs 542
Create Acc Dir 544
Install EAS 546
Setup Scheduler
548
Connect to EMS
Client 550
Setup 552
Schema 554
Files 556
Rules 558
Data 560

Figure 5

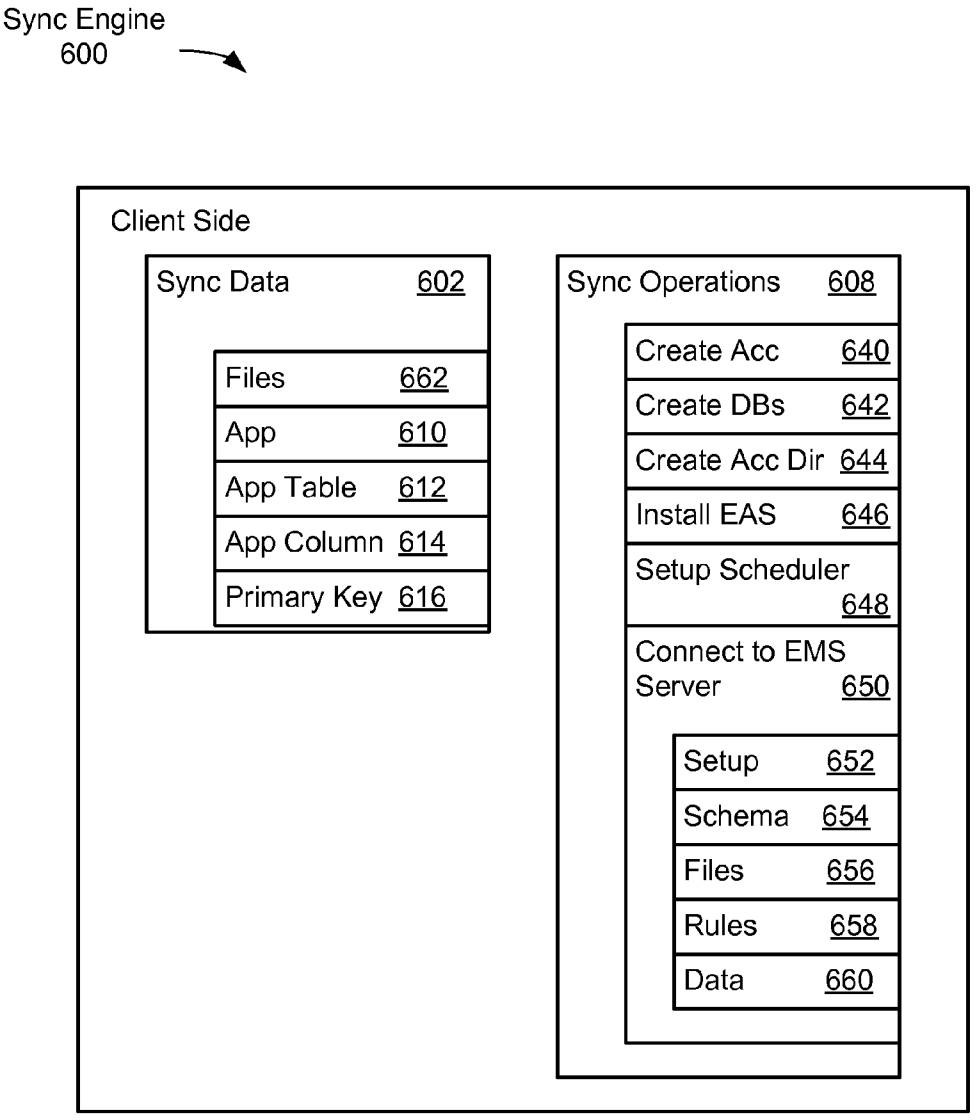

Sync Engine
600
Client Side
Sync Data 602
Files 662
App 610
App Table 612
App Column 614
Primary Key 616
Sync Operations 608
Create Acc 640
Create DBs 642
Create Acc Dir 644
Install EAS 646
Setup Scheduler 648
Connect to EMS Server 650
Setup 652
Schema 654
Files 656
Rules 658
Data 660

Figure 6

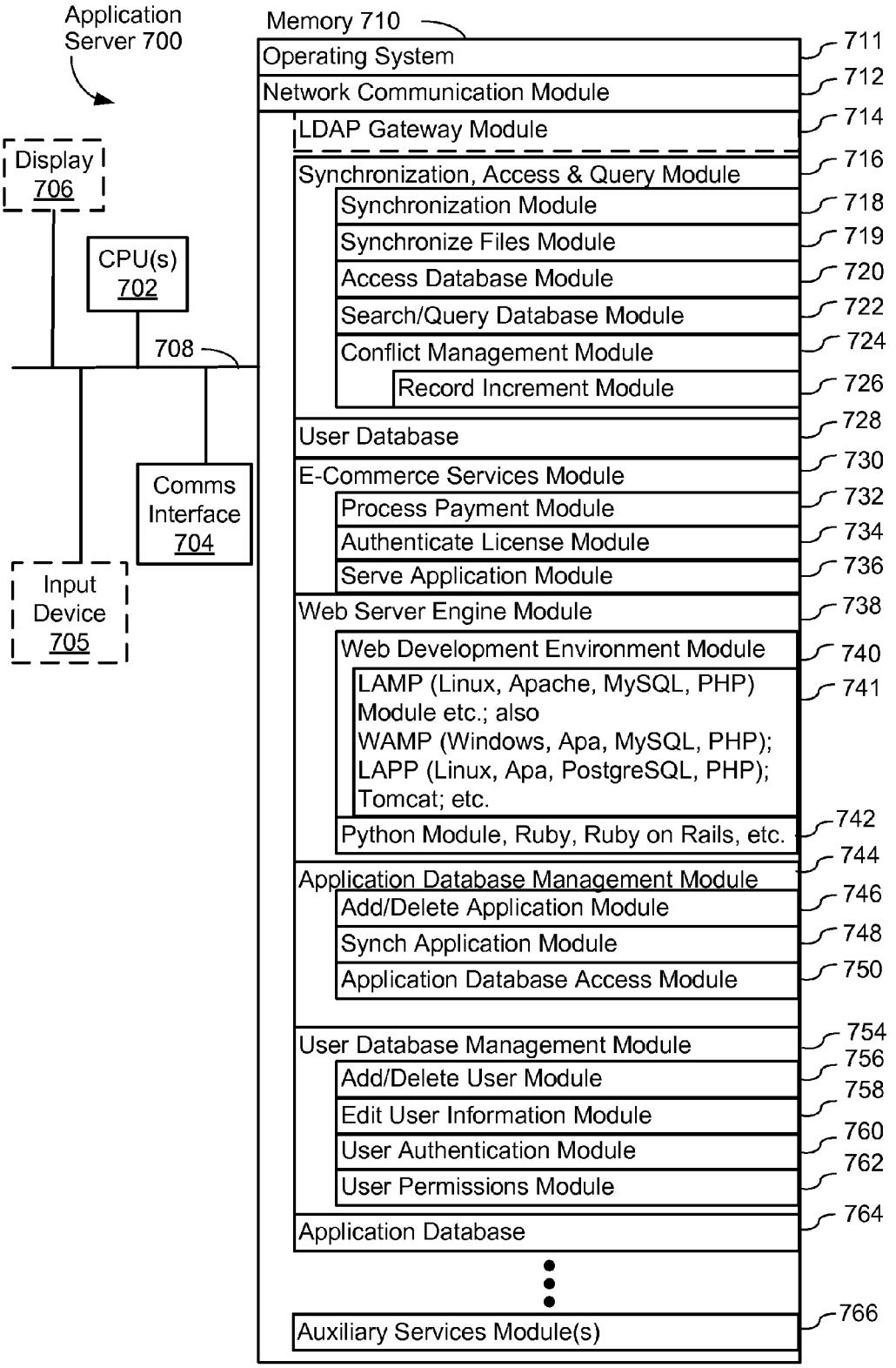

Application Server 700
Memory 710
Operating System
Network Communication Module
LDAP Gateway Module
Synchronization, Access & Query Module
Synchronization Module
Synchronize Files Module
Access Database Module
Search/Query Database Module
Conflict Management Module
Record Increment Module
User Database
E-Commerce Services Module
Process Payment Module
Authenticate License Module
Serve Application Module
Web Server Engine Module
Web Development Environment Module
LAMP (Linux, Apache, MySQL, PHP) Module etc.; also WAMP (Windows, Apa, MySQL, PHP); LAPP (Linux, Apa, PostgreSQL, PHP); Tomcat; etc.
Python Module, Ruby, Ruby on Rails, etc.
Application Database Management Module
Add/Delete Application Module
Synch Application Module
Application Database Access Module
User Database Management Module
Add/Delete User Module
Edit User Information Module
User Authentication Module
User Permissions Module
Application Database
Auxiliary Services Module(s)
711
712
714
716
718
719
720
722
724
726
728
730
732
734
736
738
740
741
742
744
746
748
750
754
756
758
760
762
764
766
Display 706
CPU(s) 702
708
Comms Interface 704
Input Device 705

Figure 7

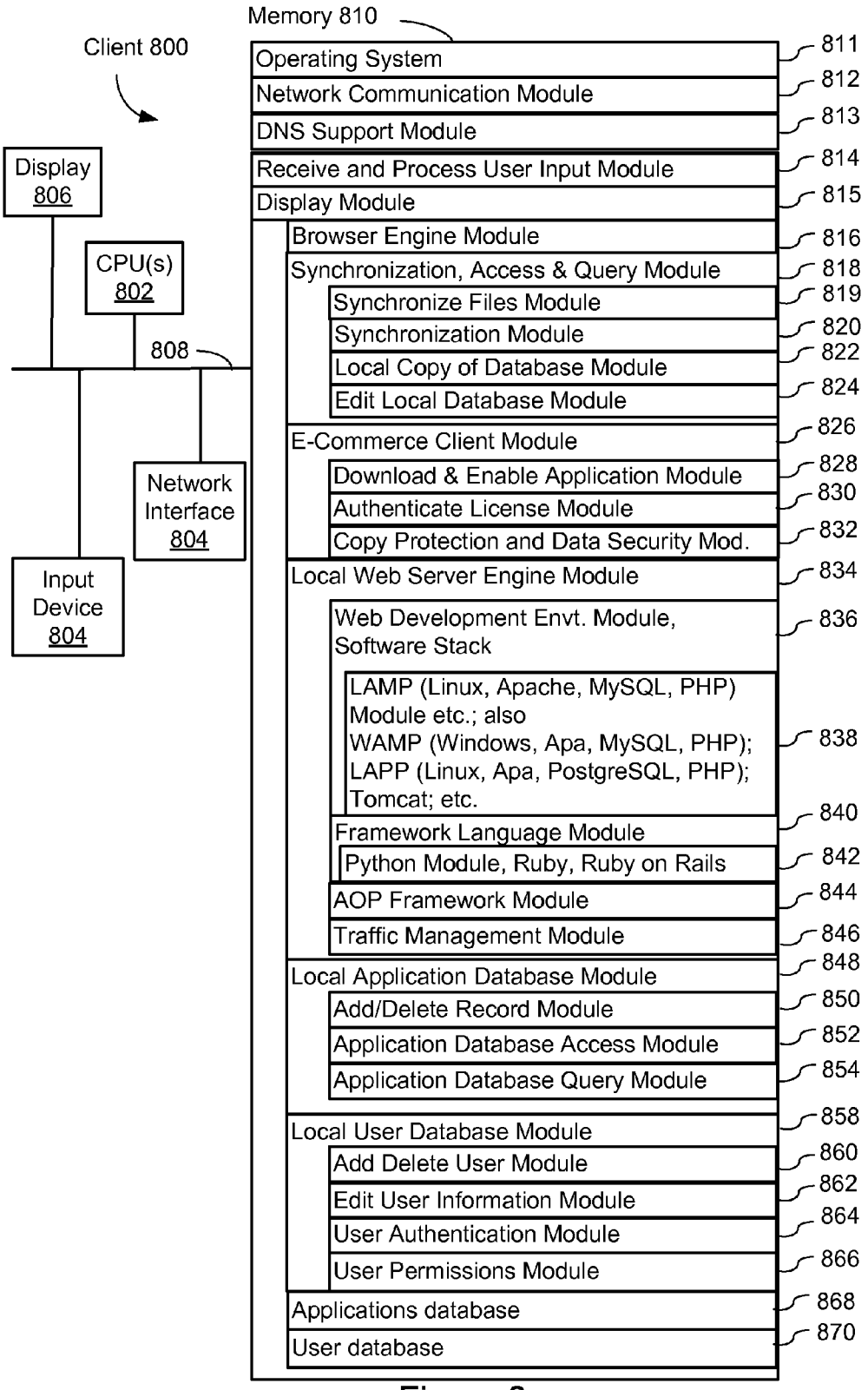

Client 800
Memory 810
Display 806
CPU(s) 802
808
Network Interface 804
Input Device 804
Operating System
Network Communication Module
DNS Support Module
Receive and Process User Input Module
Display Module
Browser Engine Module
Synchronization, Access & Query Module
Synchronize Files Module
Synchronization Module
Local Copy of Database Module
Edit Local Database Module
E-Commerce Client Module
Download & Enable Application Module
Authenticate License Module
Copy Protection and Data Security Mod.
Local Web Server Engine Module
Web Development Envt. Module, Software Stack
LAMP (Linux, Apache, MySQL, PHP) Module etc.; also WAMP (Windows, Apa, MySQL, PHP); LAPP (Linux, Apa, PostgreSQL, PHP); Tomcat; etc.
Framework Language Module
Python Module, Ruby, Ruby on Rails
AOP Framework Module
Traffic Management Module
Local Application Database Module
Add/Delete Record Module
Application Database Access Module
Application Database Query Module
Local User Database Module
Add Delete User Module
Edit User Information Module
User Authentication Module
User Permissions Module
Applications database
User database
811
812
813
814
815
816
818
819
820
822
824
826
828
830
832
834
836
838
840
842
844
846
848
850
852
854
858
860
862
864
866
868
870

Figure 8

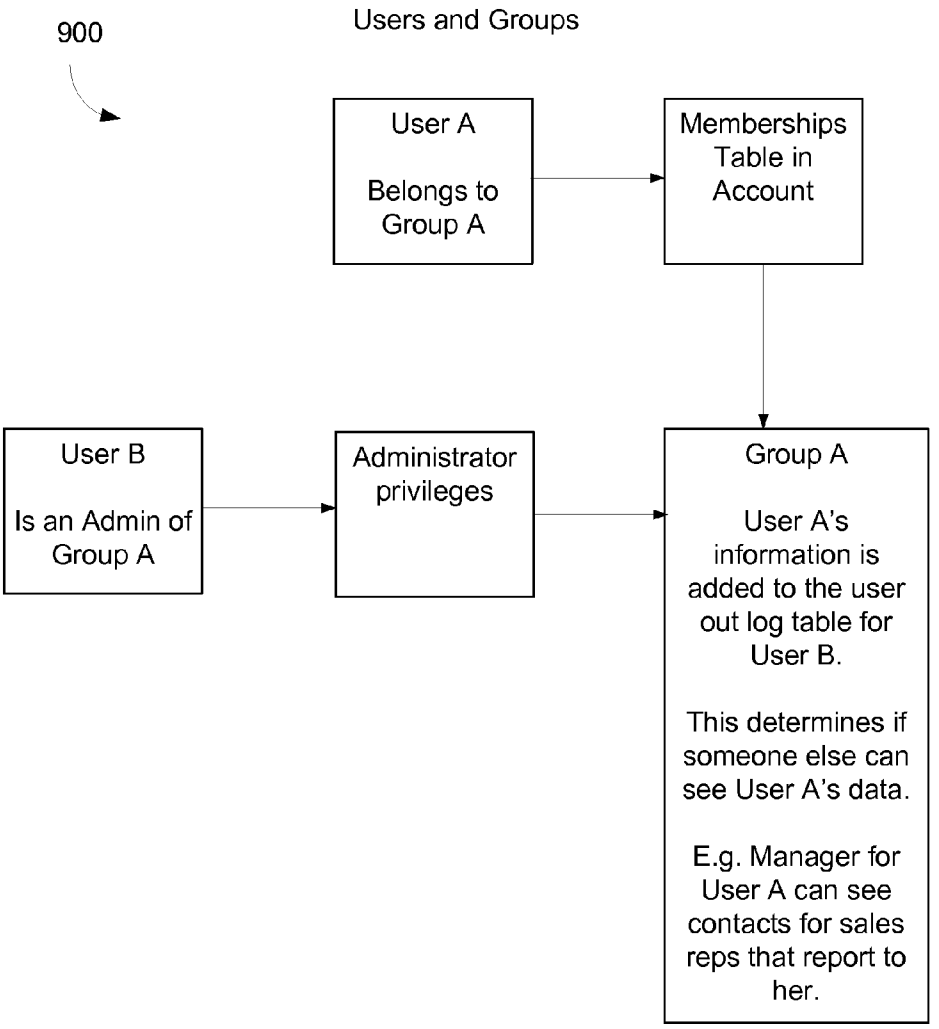

900
Users and Groups
User A
Belongs to Group A
Memberships Table in Account
User B
Is an Admin of Group A
Administrator privileges
Group A
User A's information is added to the user out log table for User B.
This determines if someone else can see User A's data.
E.g. Manager for User A can see contacts for sales reps that report to her.

Figure 9

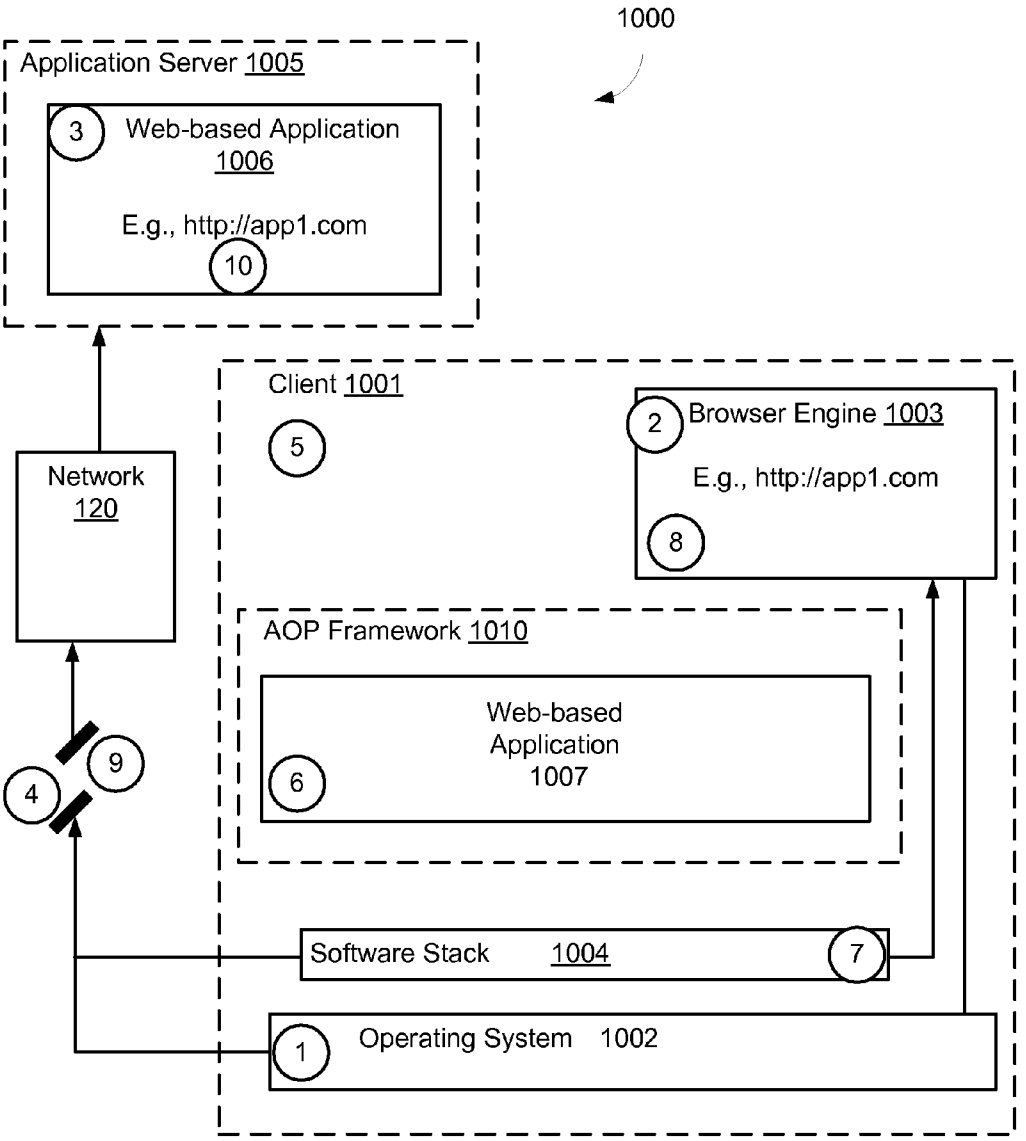

1000
Application Server 1005
3
Web-based Application 1006
E.g., http://app1.com
10
Client 1001
5
2
Browser Engine 1003
E.g., http://app1.com
8
Network 120
AOP Framework 1010
Web-based Application 1007
6
9
4
Software Stack 1004
7
1
Operating System 1002

Figure 10

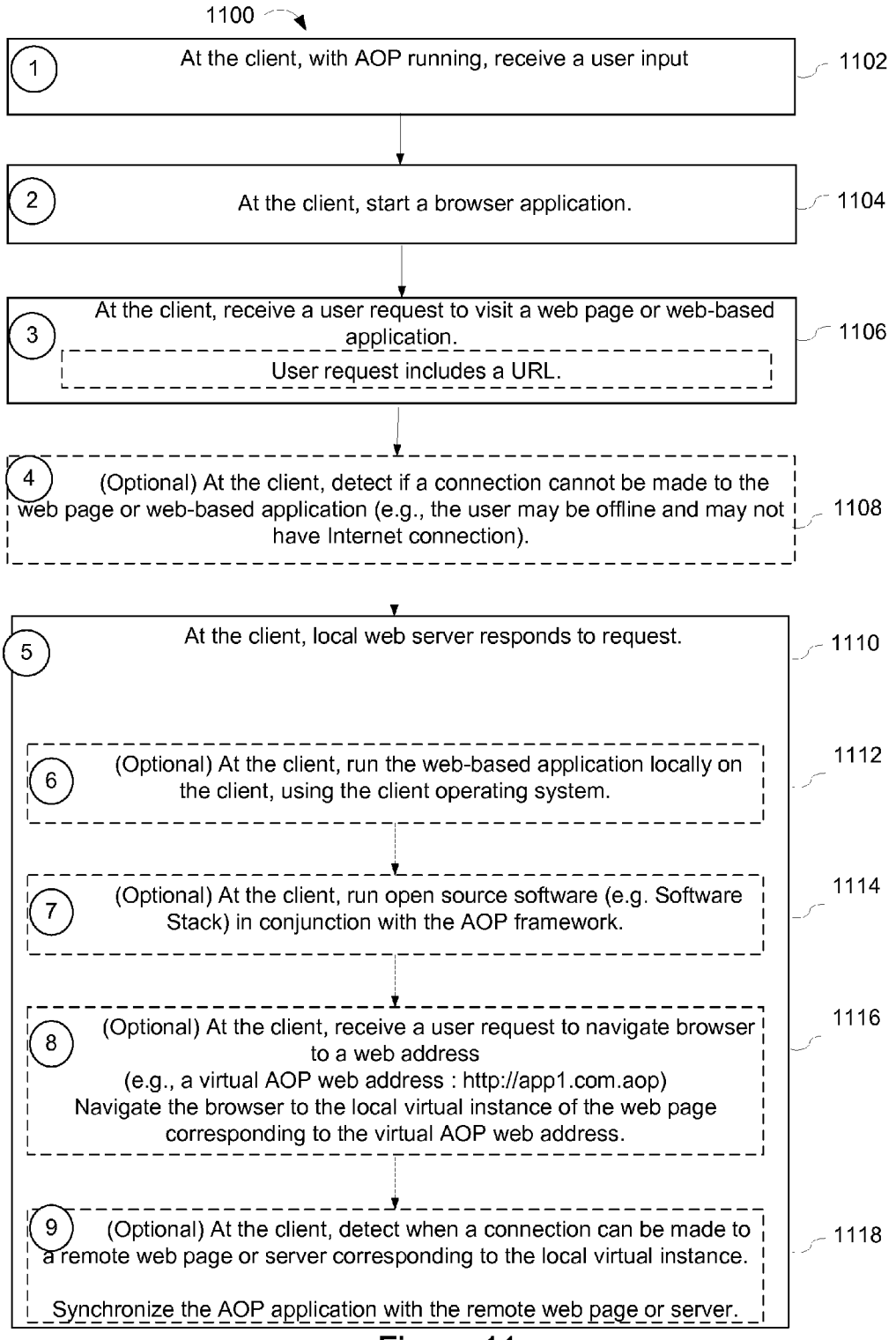

1100
1
At the client, with AOP running, receive a user input
1102
2
At the client, start a browser application.
1104
3
At the client, receive a user request to visit a web page or web-based application.
User request includes a URL.
1106
4
(Optional) At the client, detect if a connection cannot be made to the web page or web-based application (e.g., the user may be offline and may not have Internet connection).
1108
5
At the client, local web server responds to request.
1110
6
(Optional) At the client, run the web-based application locally on the client, using the client operating system.
1112
7
(Optional) At the client, run open source software (e.g. Software Stack) in conjunction with the AOP framework.
1114
8
(Optional) At the client, receive a user request to navigate browser to a web address
(e.g., a virtual AOP web address : http://app1.com.aop)
Navigate the browser to the local virtual instance of the web page corresponding to the virtual AOP web address.
1116
9
(Optional) At the client, detect when a connection can be made to a remote web page or server corresponding to the local virtual instance.
Synchronize the AOP application with the remote web page or server.
1118

Figure 11

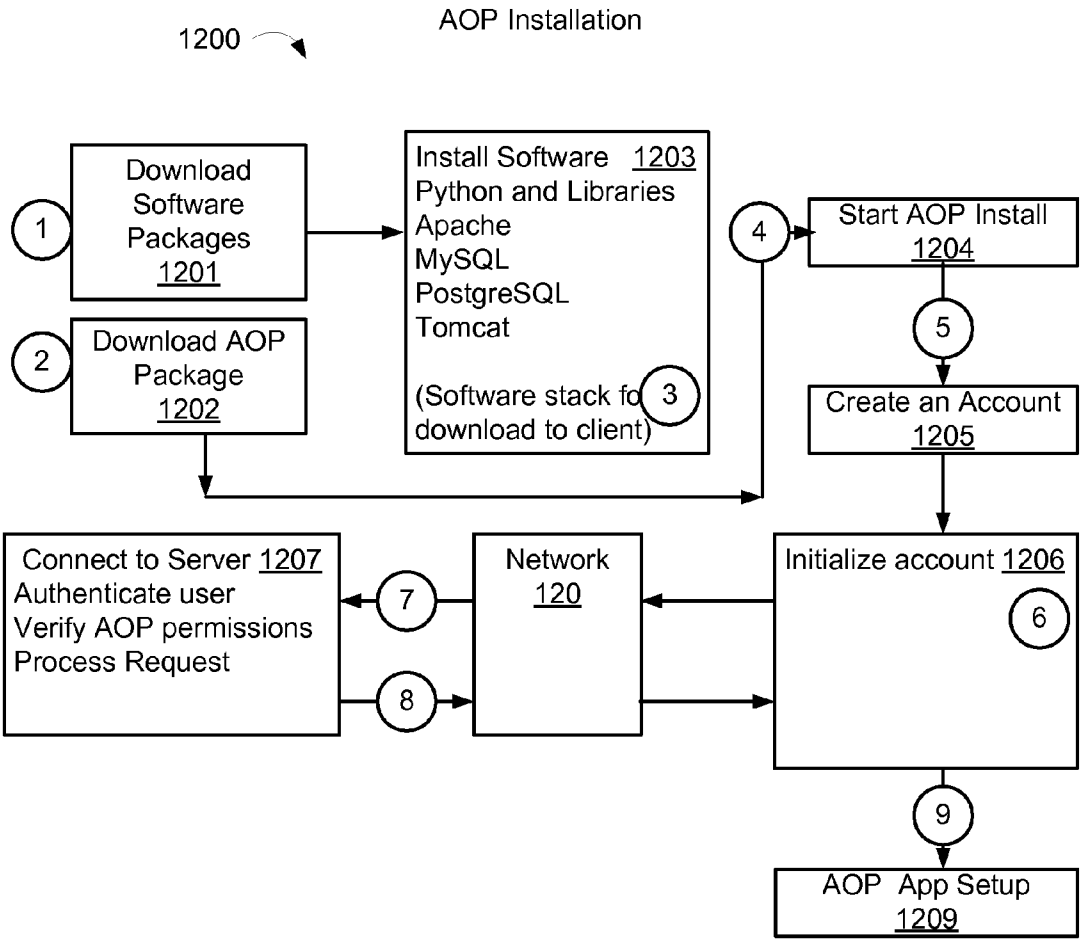

AOP Installation
1200
Download Software Packages 1201
Download AOP Package 1202
Install Software 1203
Python and Libraries
Apache
MySQL
PostgreSQL
Tomcat
(Software stack fo download to client)
Start AOP Install 1204
Create an Account 1205
Initialize account 1206
Network 120
Connect to Server 1207
Authenticate user
Verify AOP permissions
Process Request
AOP App Setup 1209
1
2
3
4
5
6
7
8
9

Figure 12

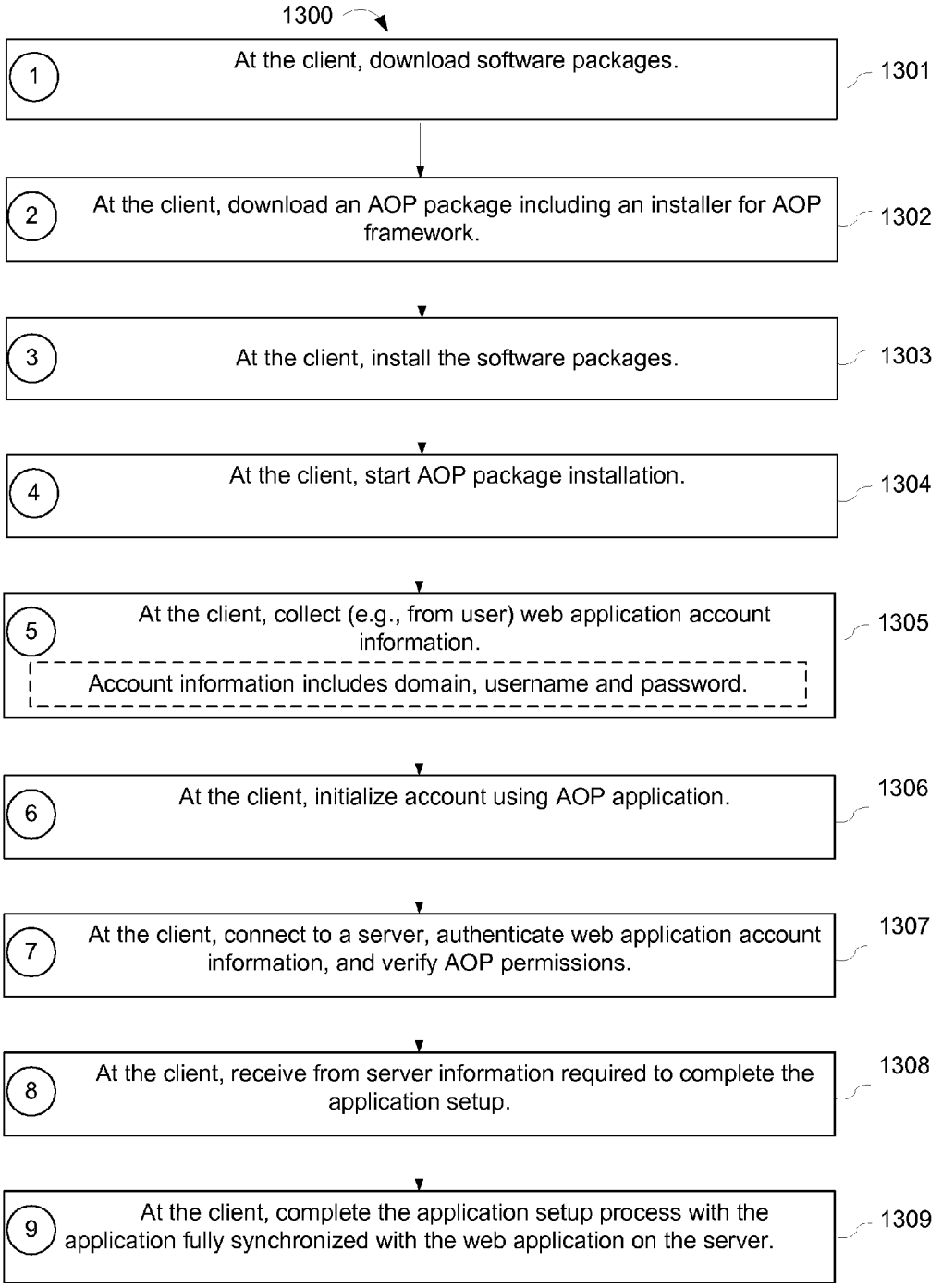

1300
1
At the client, download software packages.
1301
2
At the client, download an AOP package including an installer for AOP framework.
1302
3
At the client, install the software packages.
1303
4
At the client, start AOP package installation.
1304
5
At the client, collect (e.g., from user) web application account information.
Account information includes domain, username and password.
1305
6
At the client, initialize account using AOP application.
1306
7
At the client, connect to a server, authenticate web application account information, and verify AOP permissions.
1307
8
At the client, receive from server information required to complete the application setup.
1308
9
At the client, complete the application setup process with the application fully synchronized with the web application on the server.
1309

Figure 13

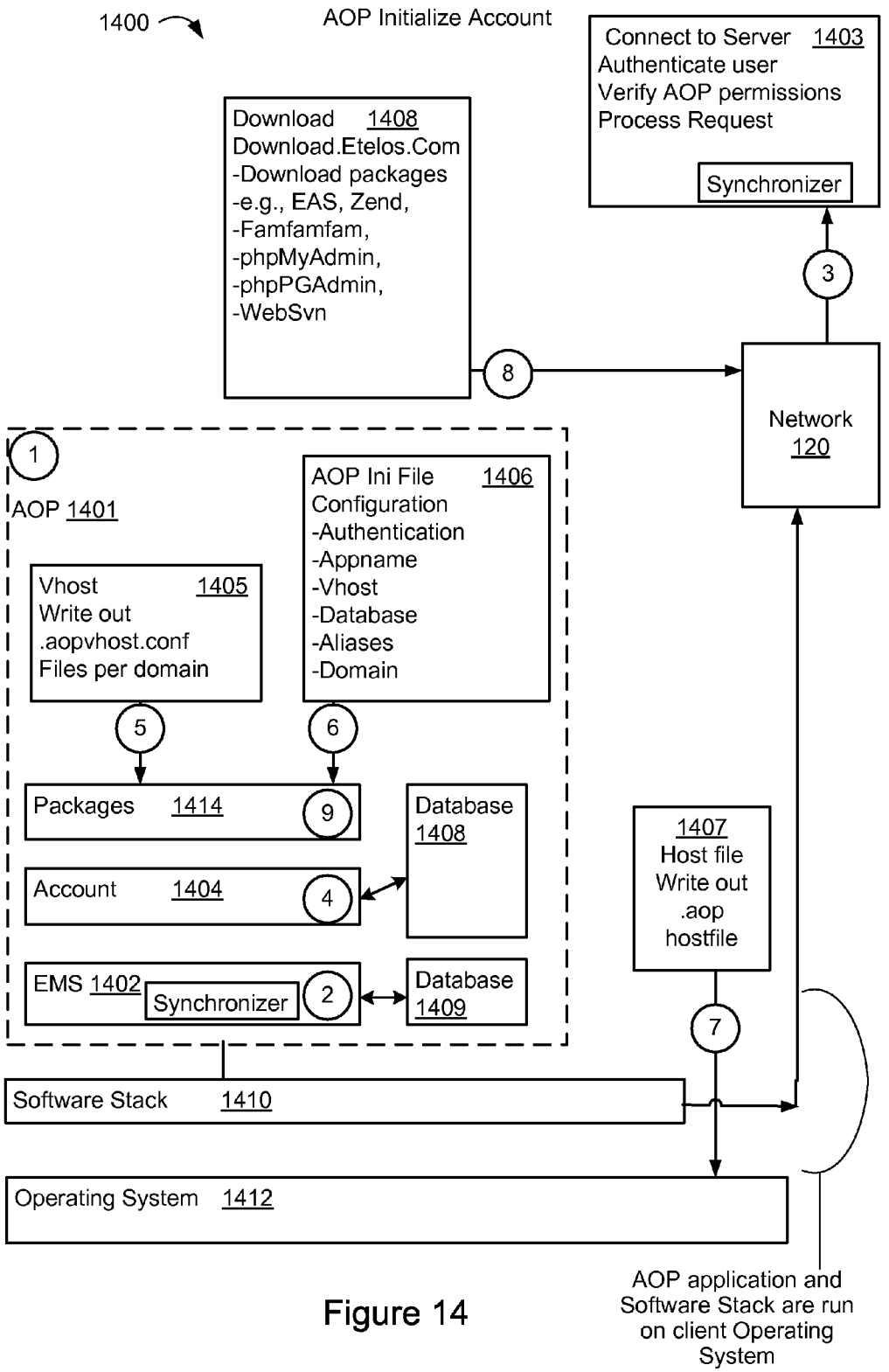

1400
AOP Initialize Account
Connect to Server 1403
Authenticate user
Verify AOP permissions
Process Request
Synchronizer
Download 1408
Download.Etelos.Com
-Download packages
-e.g., EAS, Zend,
-Famfamfam,
-phpMyAdmin,
-phpPGAdmin,
-WebSvn
3
8
Network
120
1
AOP 1401
AOP Ini File 1406
Configuration
-Authentication
-Appname
-Vhost
-Database
-Aliases
-Domain
Vhost 1405
Write out
.aopvhost.conf
Files per domain
5
6
Packages 1414
9
Database
1408
Account 1404
4
1407
Host file
Write out
.aop
hostfile
EMS 1402
Synchronizer
2
Database
1409
7
Software Stack 1410
Operating System 1412
AOP application and
Software Stack are run
on client Operating
System

Figure 14

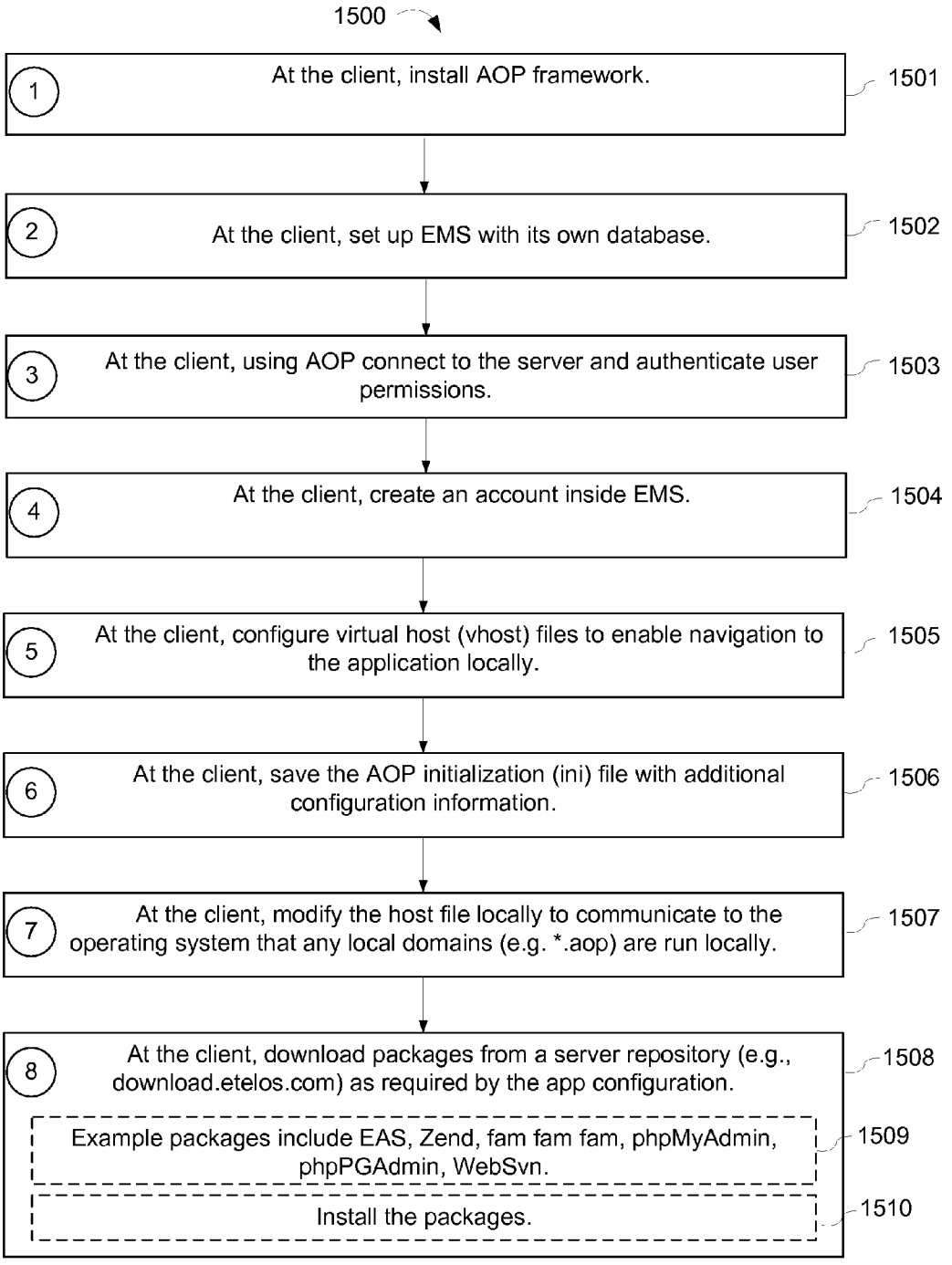

1500
1
At the client, install AOP framework.
1501
2
At the client, set up EMS with its own database.
1502
3
At the client, using AOP connect to the server and authenticate user permissions.
1503
4
At the client, create an account inside EMS.
1504
5
At the client, configure virtual host (vhost) files to enable navigation to the application locally.
1505
6
At the client, save the AOP initialization (ini) file with additional configuration information.
1506
7
At the client, modify the host file locally to communicate to the operating system that any local domains (e.g. *.aop) are run locally.
1507
8
At the client, download packages from a server repository (e.g., download.etelos.com) as required by the app configuration.
1508
Example packages include EAS, Zend, fam fam fam, phpMyAdmin, phpPGAdmin, WebSvn.
1509
Install the packages.
1510

Figure 15

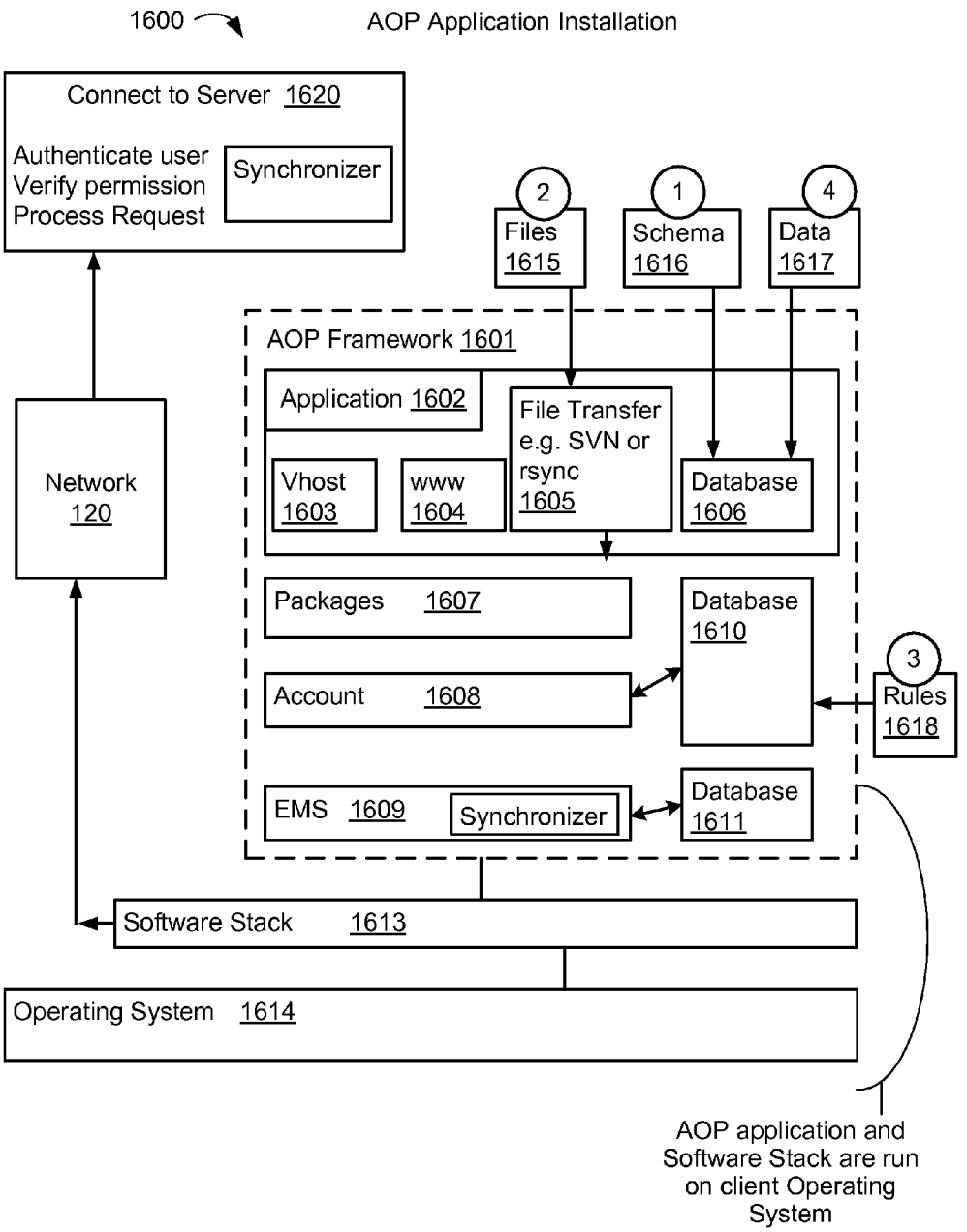

1600
AOP Application Installation
Connect to Server 1620
Authenticate user
Verify permission
Process Request
Synchronizer
2
Files 1615
1
Schema 1616
4
Data 1617
AOP Framework 1601
Application 1602
File Transfer e.g. SVN or rsync 1605
Vhost 1603
www 1604
Database 1606
Network 120
Packages 1607
Database 1610
Account 1608
3
Rules 1618
EMS 1609
Synchronizer
Database 1611
Software Stack 1613
Operating System 1614
AOP application and Software Stack are run on client Operating System

Figure 16

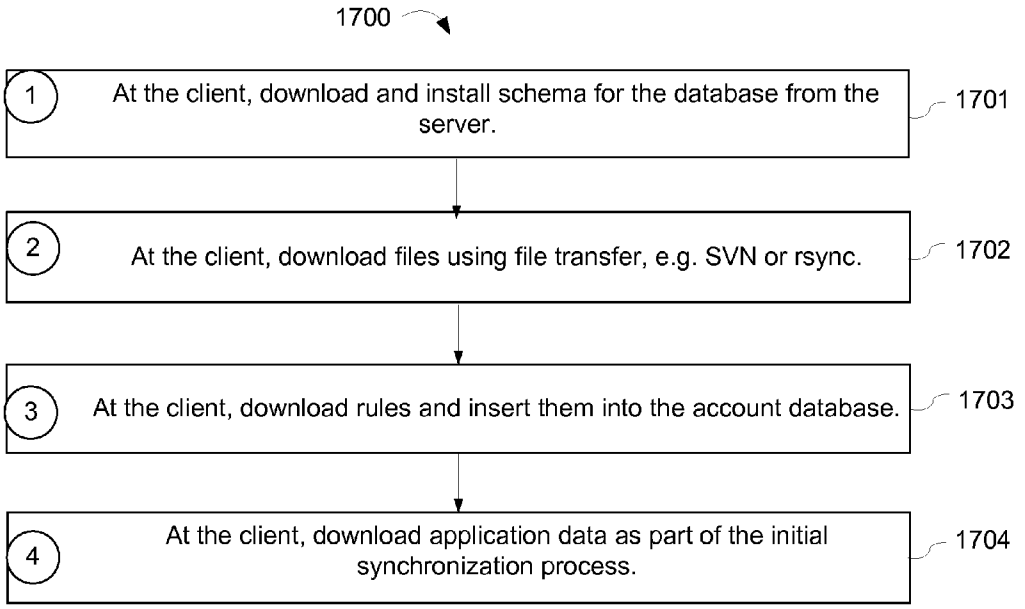

1700
1
At the client, download and install schema for the database from the server.
1701
2
At the client, download files using file transfer, e.g. SVN or rsync.
1702
3
At the client, download rules and insert them into the account database.
1703
4
At the client, download application data as part of the initial synchronization process.
1704

Figure 17

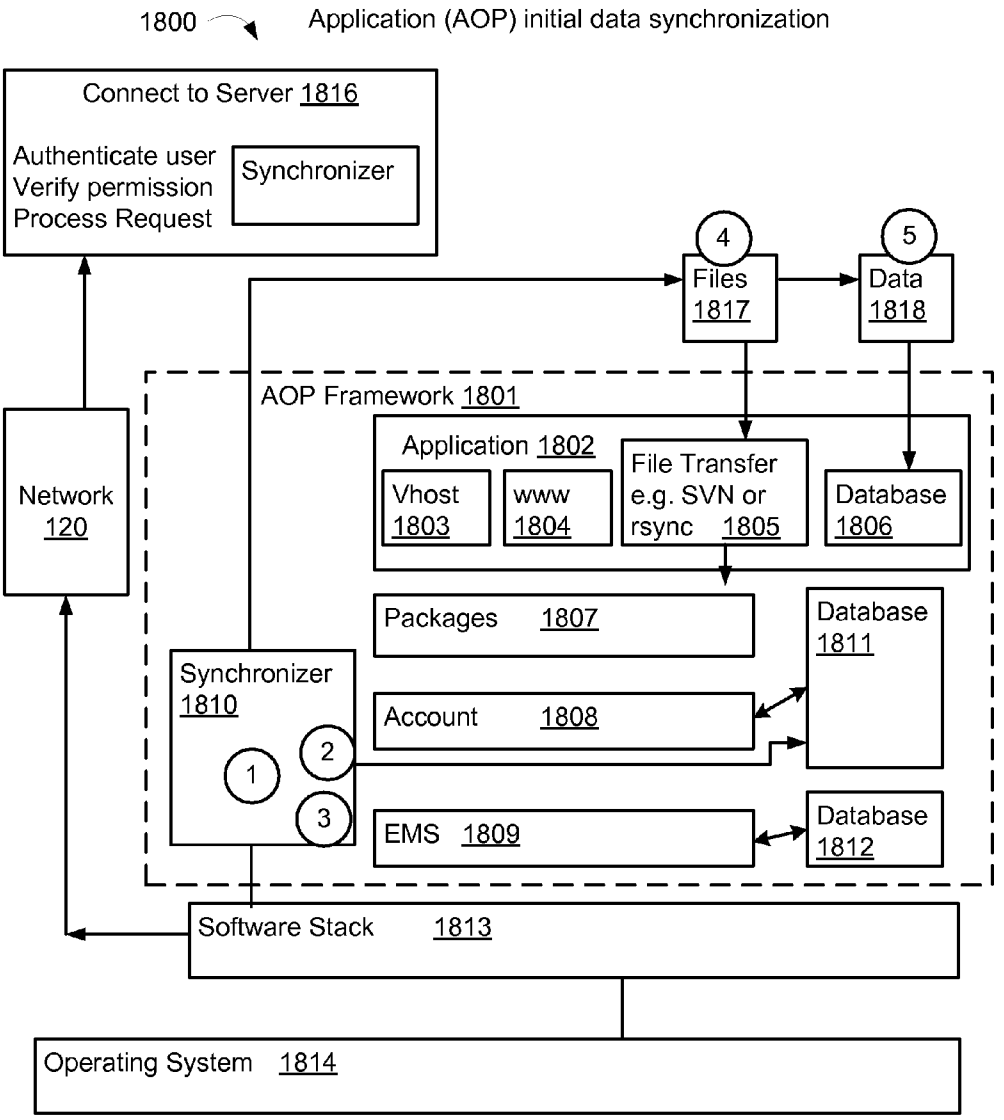

1800
Application (AOP) initial data synchronization
Connect to Server 1816
Authenticate user
Verify permission
Process Request
Synchronizer
4
Files 1817
5
Data 1818
AOP Framework 1801
Application 1802
Vhost 1803
www 1804
File Transfer e.g. SVN or rsync 1805
Database 1806
Network 120
Packages 1807
Database 1811
Synchronizer 1810
Account 1808
1
2
3
EMS 1809
Database 1812
Software Stack 1813
Operating System 1814

Figure 18

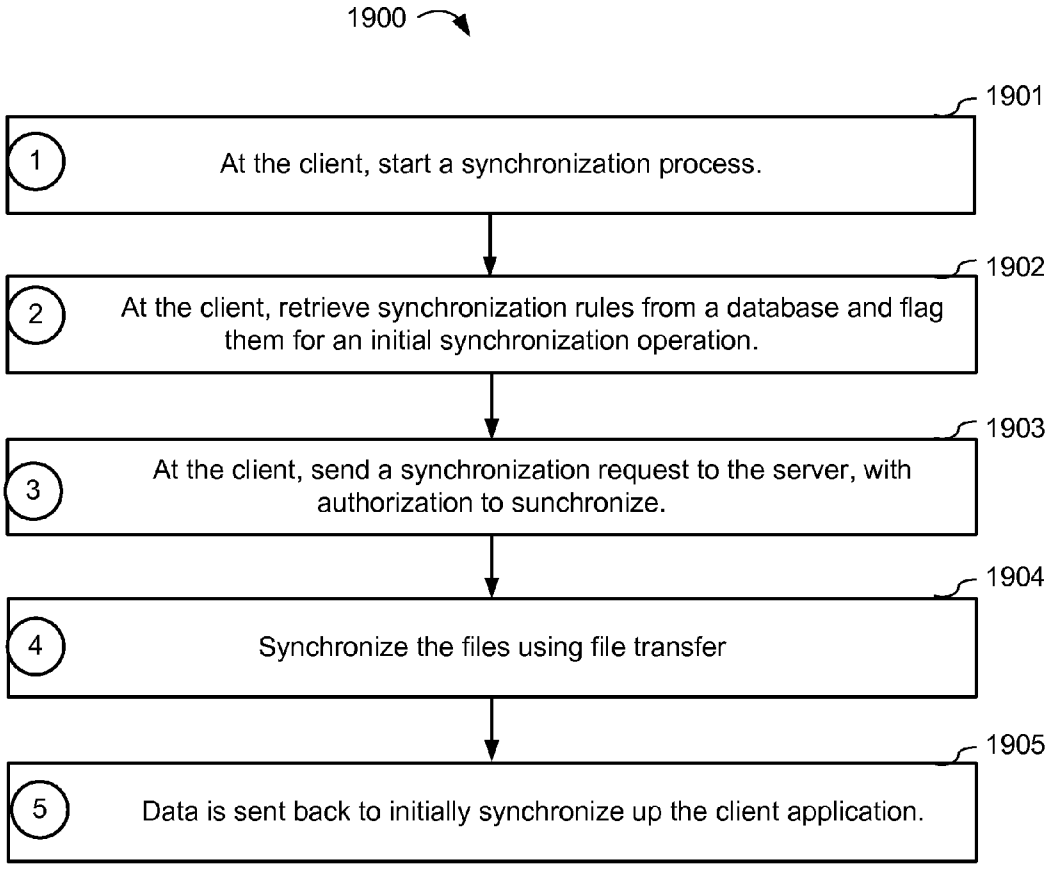

1900
1901
1
At the client, start a synchronization process.
1902
2
At the client, retrieve synchronization rules from a database and flag them for an initial synchronization operation.
1903
3
At the client, send a synchronization request to the server, with authorization to sunchronize.
1904
4
Synchronize the files using file transfer
1905
5
Data is sent back to initially synchronize up the client application.

Figure 19

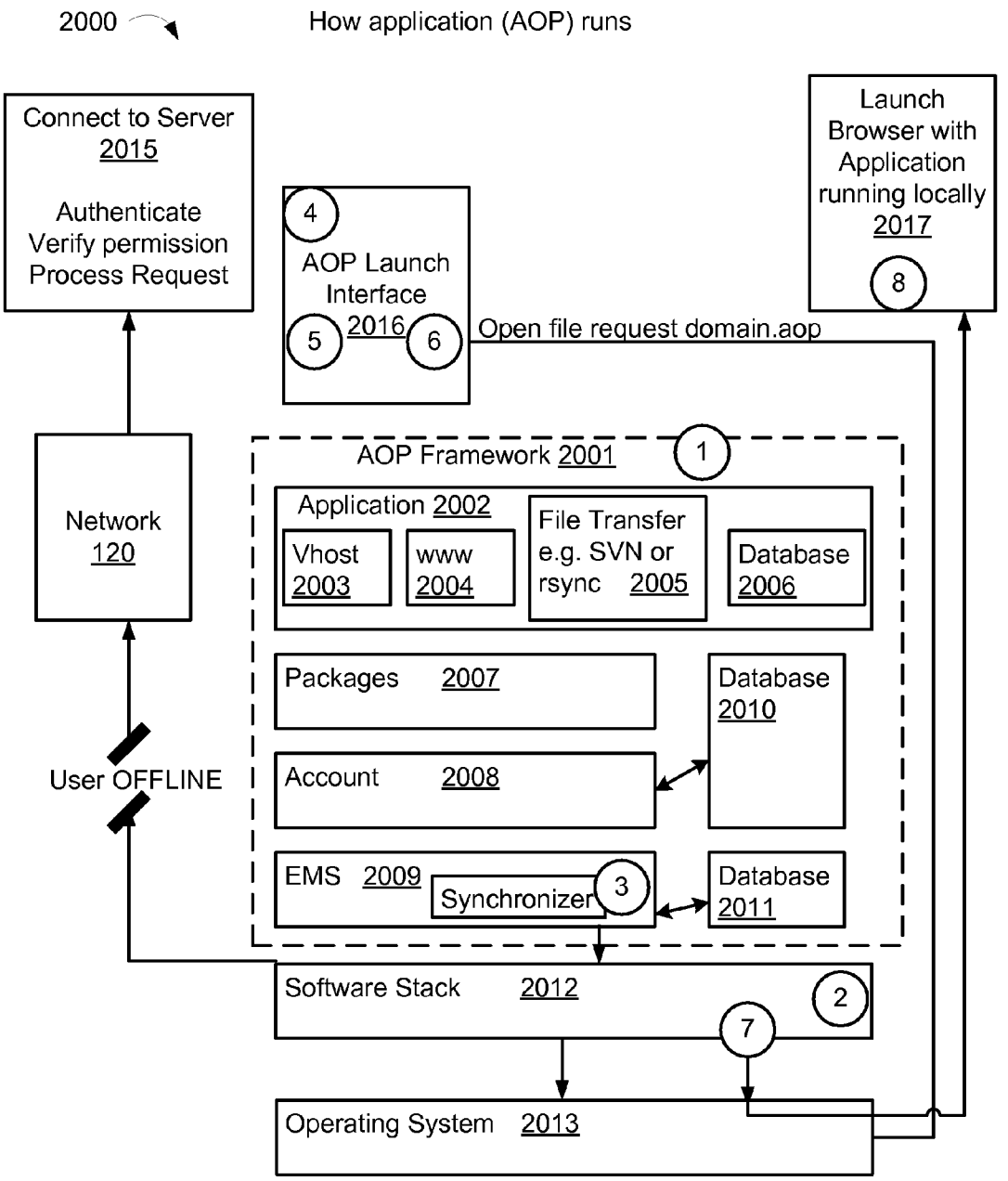

2000
How application (AOP) runs
Connect to Server 2015
Authenticate
Verify permission
Process Request
4
AOP Launch Interface
5
2016
6
Open file request domain.aop
Launch Browser with Application running locally 2017
8
Network 120
AOP Framework 2001
1
Application 2002
Vhost 2003
www 2004
File Transfer e.g. SVN or rsync 2005
Database 2006
Packages 2007
Database 2010
Account 2008
EMS 2009
Synchronizer
3
Database 2011
User OFFLINE
Software Stack 2012
2
7
Operating System 2013

Figure 20

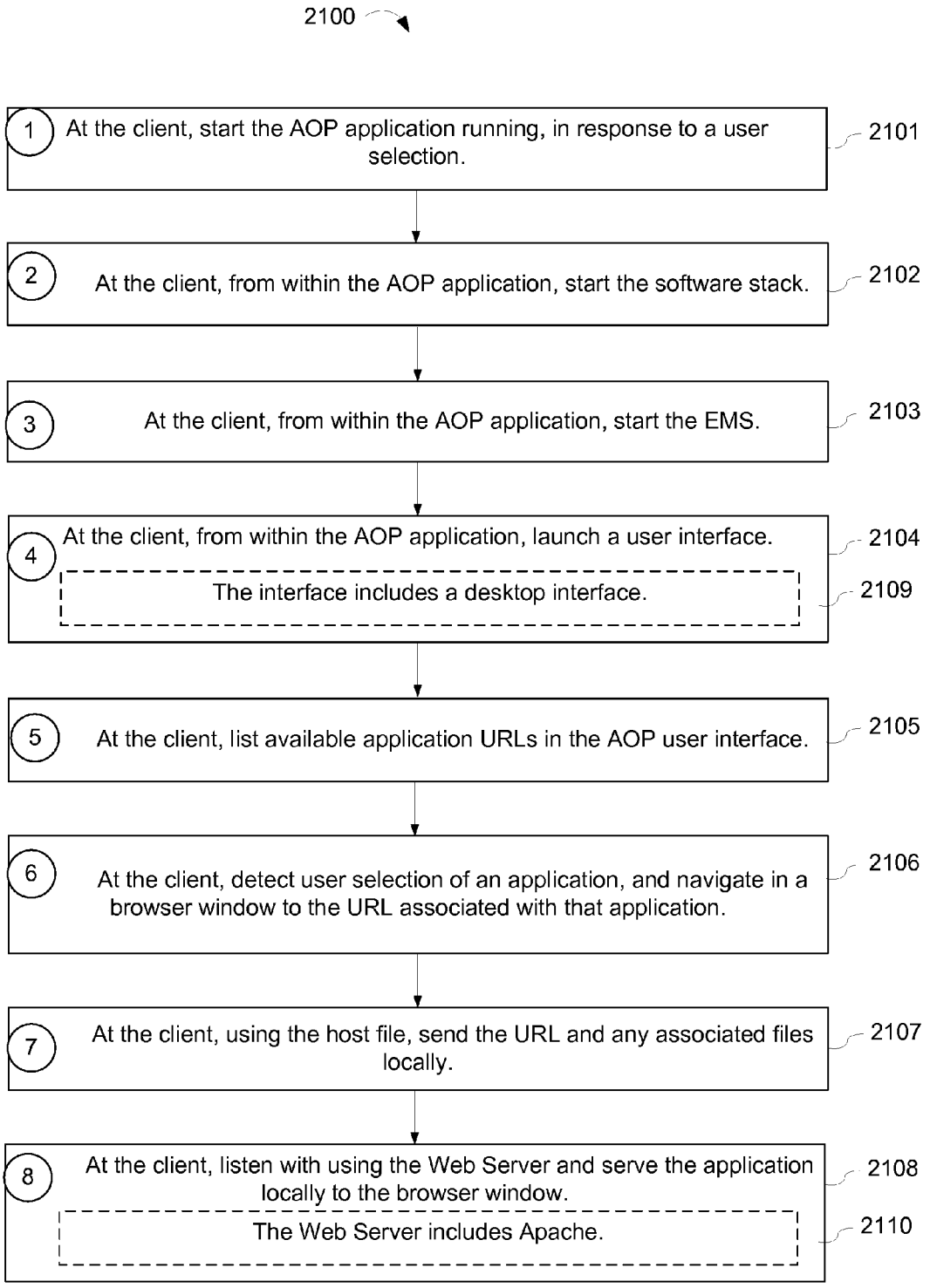

2100
1
At the client, start the AOP application running, in response to a user selection.
2101
2
At the client, from within the AOP application, start the software stack.
2102
3
At the client, from within the AOP application, start the EMS.
2103
4
At the client, from within the AOP application, launch a user interface.
2104
The interface includes a desktop interface.
2109
5
At the client, list available application URLs in the AOP user interface.
2105
6
At the client, detect user selection of an application, and navigate in a browser window to the URL associated with that application.
2106
7
At the client, using the host file, send the URL and any associated files locally.
2107
8
At the client, listen with using the Web Server and serve the application locally to the browser window.
2108
The Web Server includes Apache.
2110

Figure 21

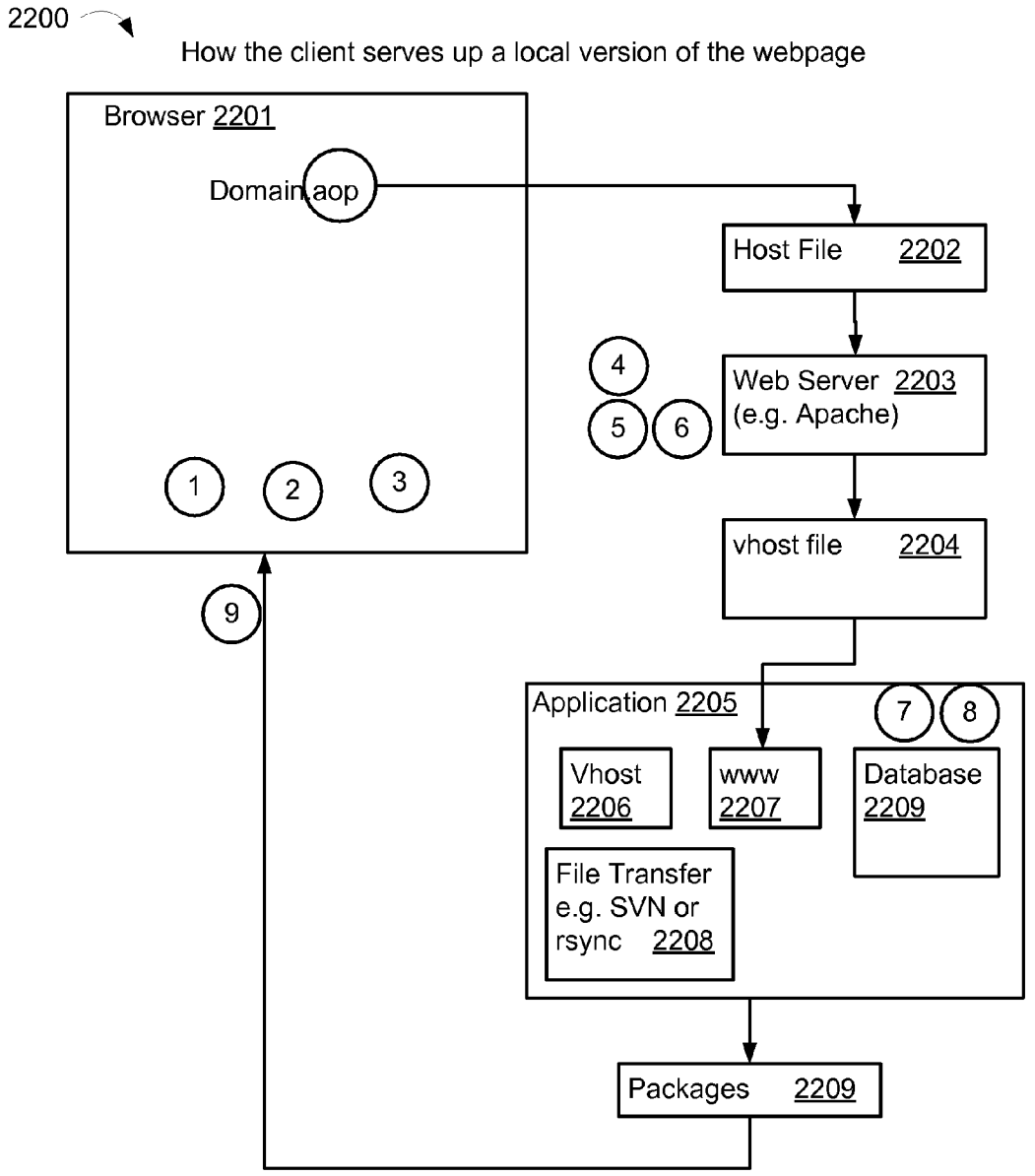

2200
How the client serves up a local version of the webpage
Browser 2201
Domain aop
Host File 2202
Web Server 2203
(e.g. Apache)
vhost file 2204
Application 2205
Vhost 2206
www 2207
Database 2209
File Transfer e.g. SVN or rsync 2208
Packages 2209
1
2
3
4
5
6
7
8
9

Figure 22

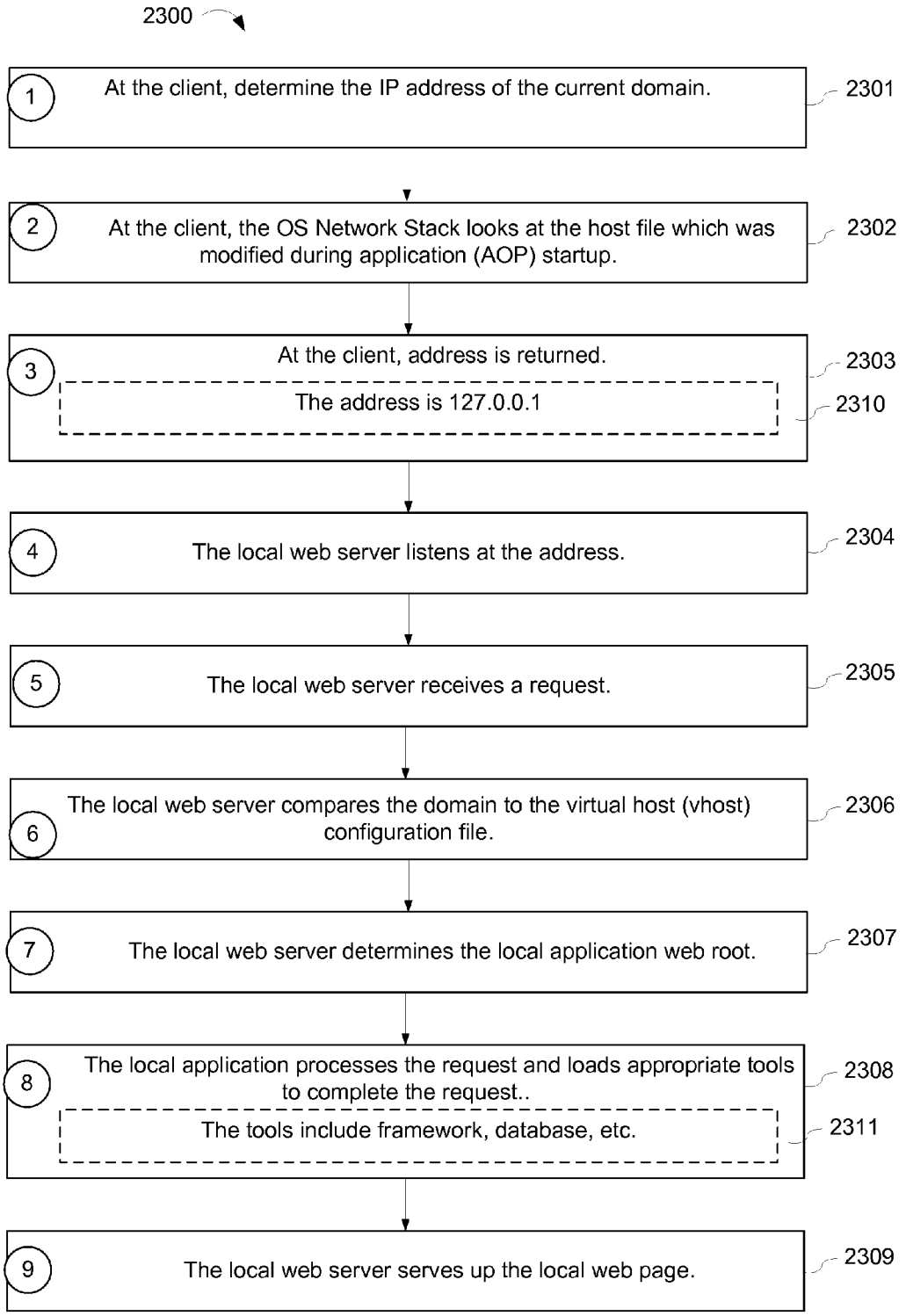

2300
1
At the client, determine the IP address of the current domain.
2301
2
At the client, the OS Network Stack looks at the host file which was modified during application (AOP) startup.
2302
3
At the client, address is returned.
The address is 127.0.0.1
2303
2310
4
The local web server listens at the address.
2304
5
The local web server receives a request.
2305
6
The local web server compares the domain to the virtual host (vhost) configuration file.
2306
7
The local web server determines the local application web root.
2307
8
The local application processes the request and loads appropriate tools to complete the request..
The tools include framework, database, etc.
2308
2311
9
The local web server serves up the local web page.
2309

Figure 23

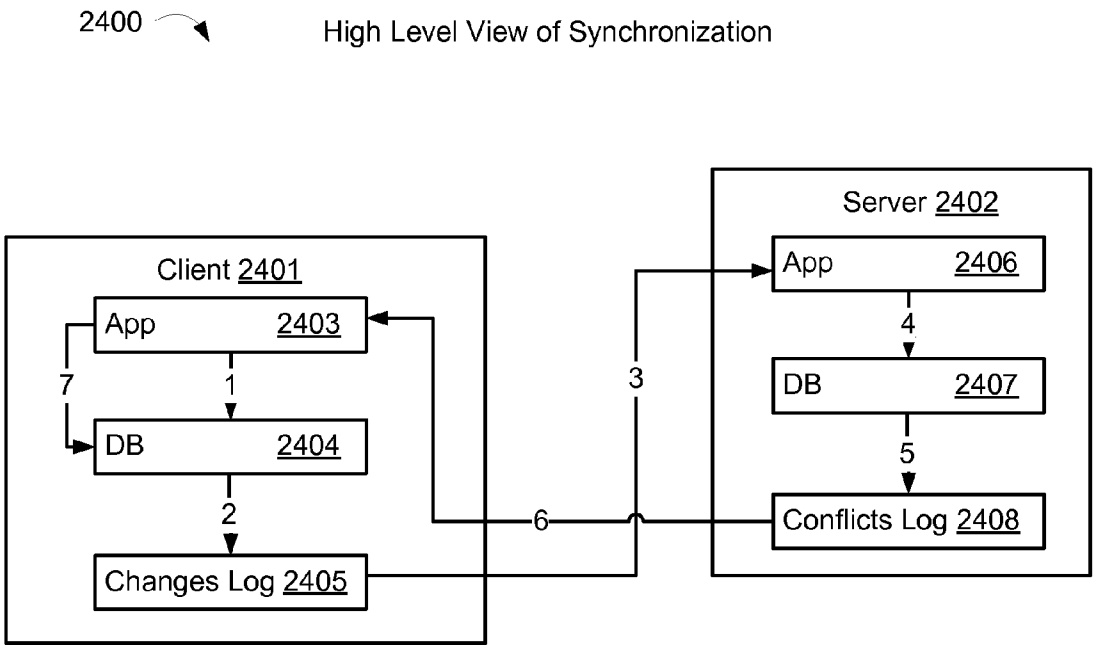

2400
High Level View of Synchronization
Server 2402
App 2406
4
DB 2407
5
Conflicts Log 2408
Client 2401
App 2403
7
1
DB 2404
2
Changes Log 2405
6
3

Figure 24

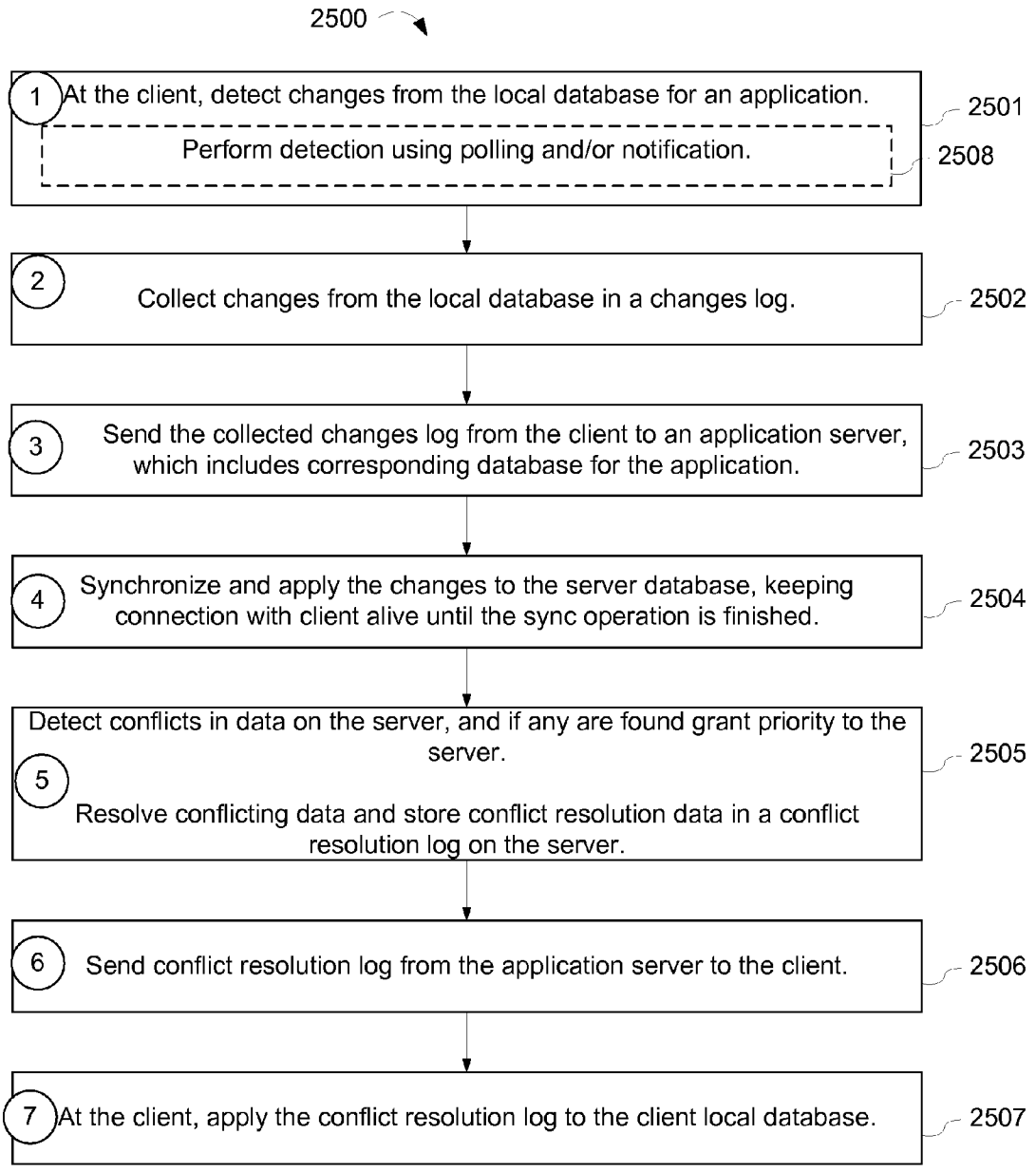

2500
1 At the client, detect changes from the local database for an application.
Perform detection using polling and/or notification.
2501
2508
2 Collect changes from the local database in a changes log.
2502
3 Send the collected changes log from the client to an application server, which includes corresponding database for the application.
2503
4 Synchronize and apply the changes to the server database, keeping connection with client alive until the sync operation is finished.
2504
Detect conflicts in data on the server, and if any are found grant priority to the server.
5 Resolve conflicting data and store conflict resolution data in a conflict resolution log on the server.
2505
6 Send conflict resolution log from the application server to the client.
2506
7 At the client, apply the conflict resolution log to the client local database.
2507

Figure 25

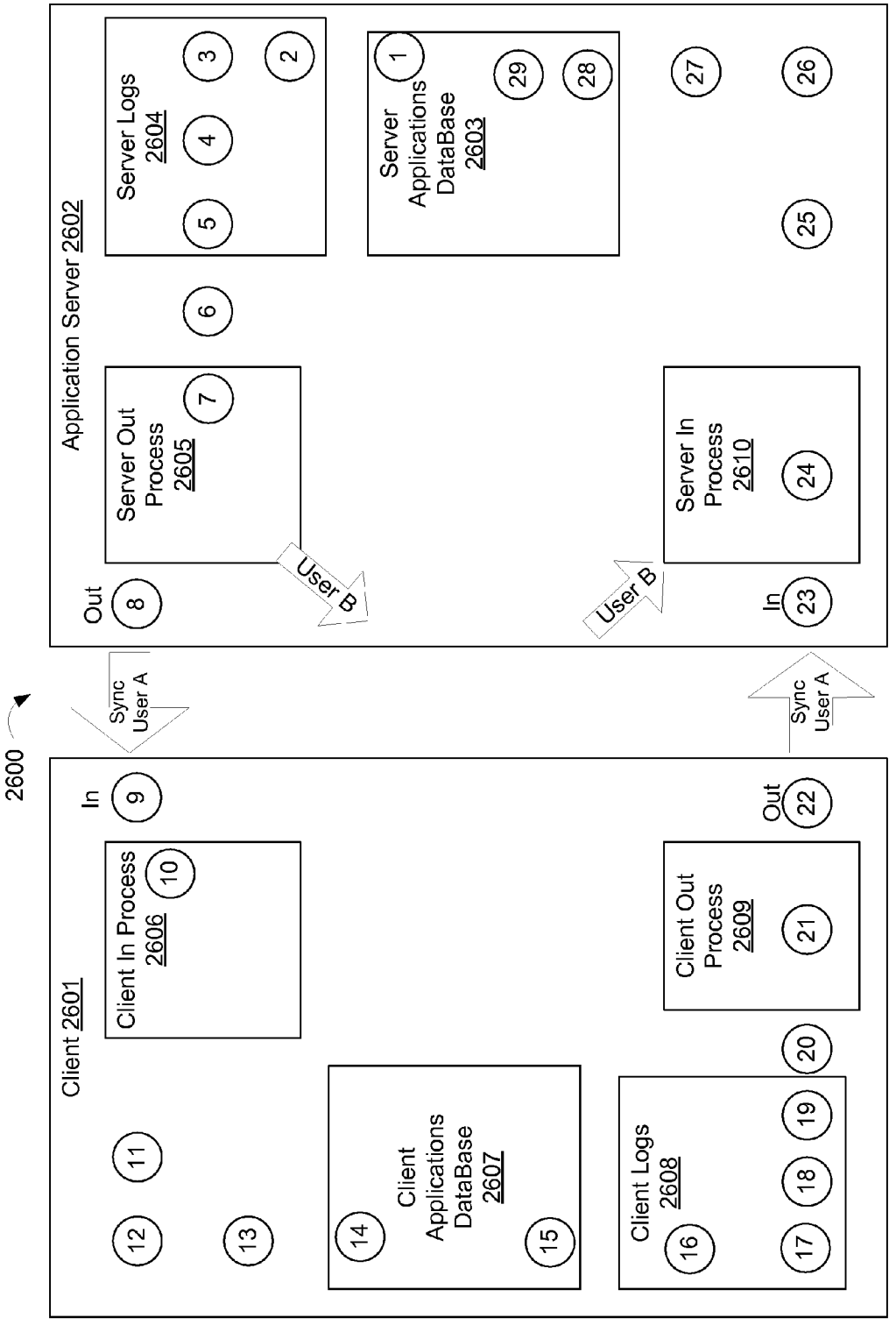

2600
Client 2601
In
9
Client In Process 2606
10
11
12
13
14
Client Applications DataBase 2607
15
16
Client Logs 2608
17
18
19
20
Client Out Process 2609
21
Out
22
Sync User A
Application Server 2602
Out
8
Server Out Process 2605
7
6
Server Logs 2604
5
4
3
2
User B
Server Applications DataBase 2603
1
29
28
27
26
25
User B
Server In Process 2610
24
In
23
Sync User A

Figure 26

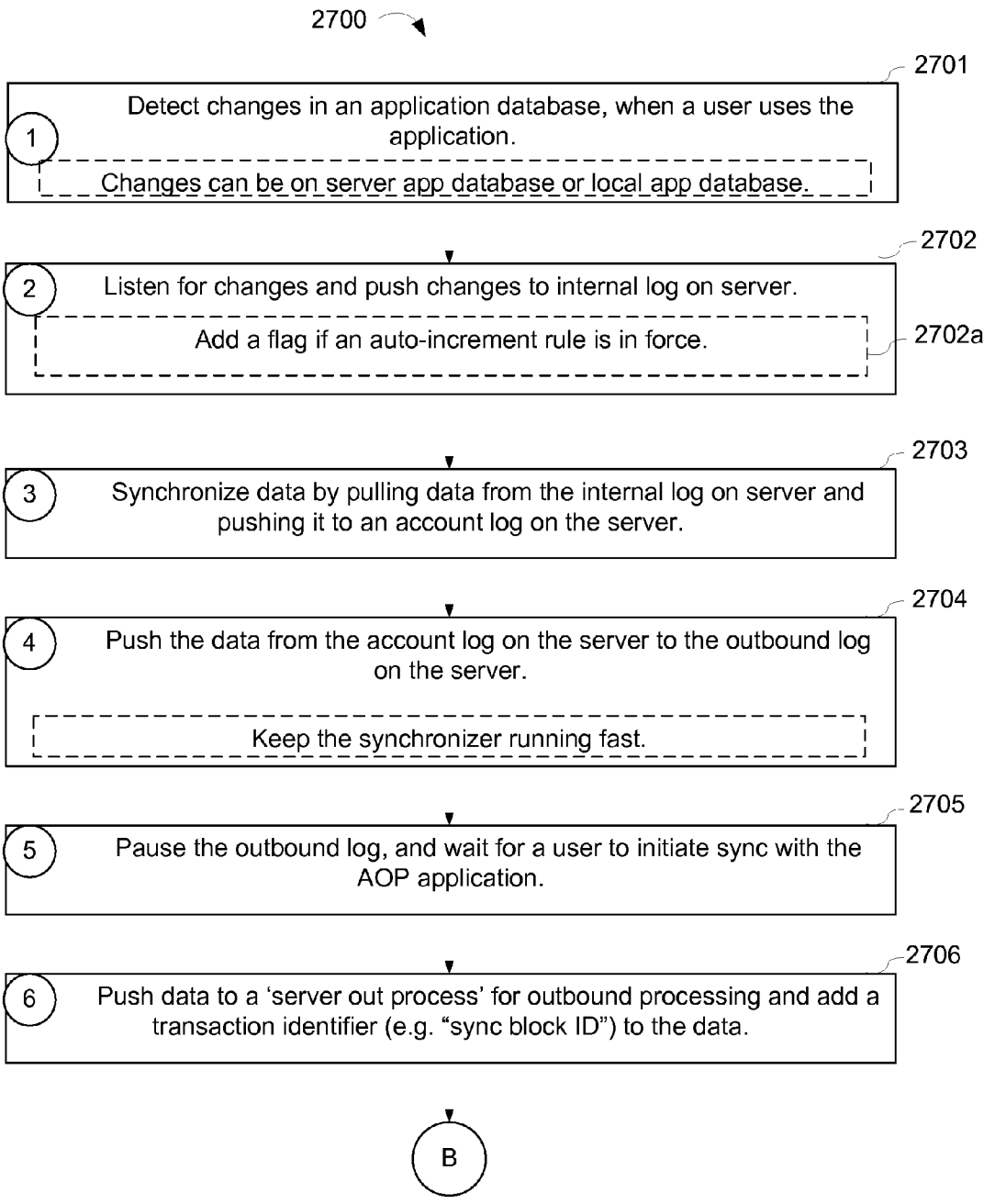

2700
2701
Detect changes in an application database, when a user uses the application.
1
Changes can be on server app database or local app database.
2702
2
Listen for changes and push changes to internal log on server.
Add a flag if an auto-increment rule is in force.
2702a
2703
3
Synchronize data by pulling data from the internal log on server and pushing it to an account log on the server.
2704
4
Push the data from the account log on the server to the outbound log on the server.
Keep the synchronizer running fast.
2705
5
Pause the outbound log, and wait for a user to initiate sync with the AOP application.
2706
6
Push data to a 'server out process' for outbound processing and add a transaction identifier (e.g. "sync block ID") to the data.
B

Figure 27A

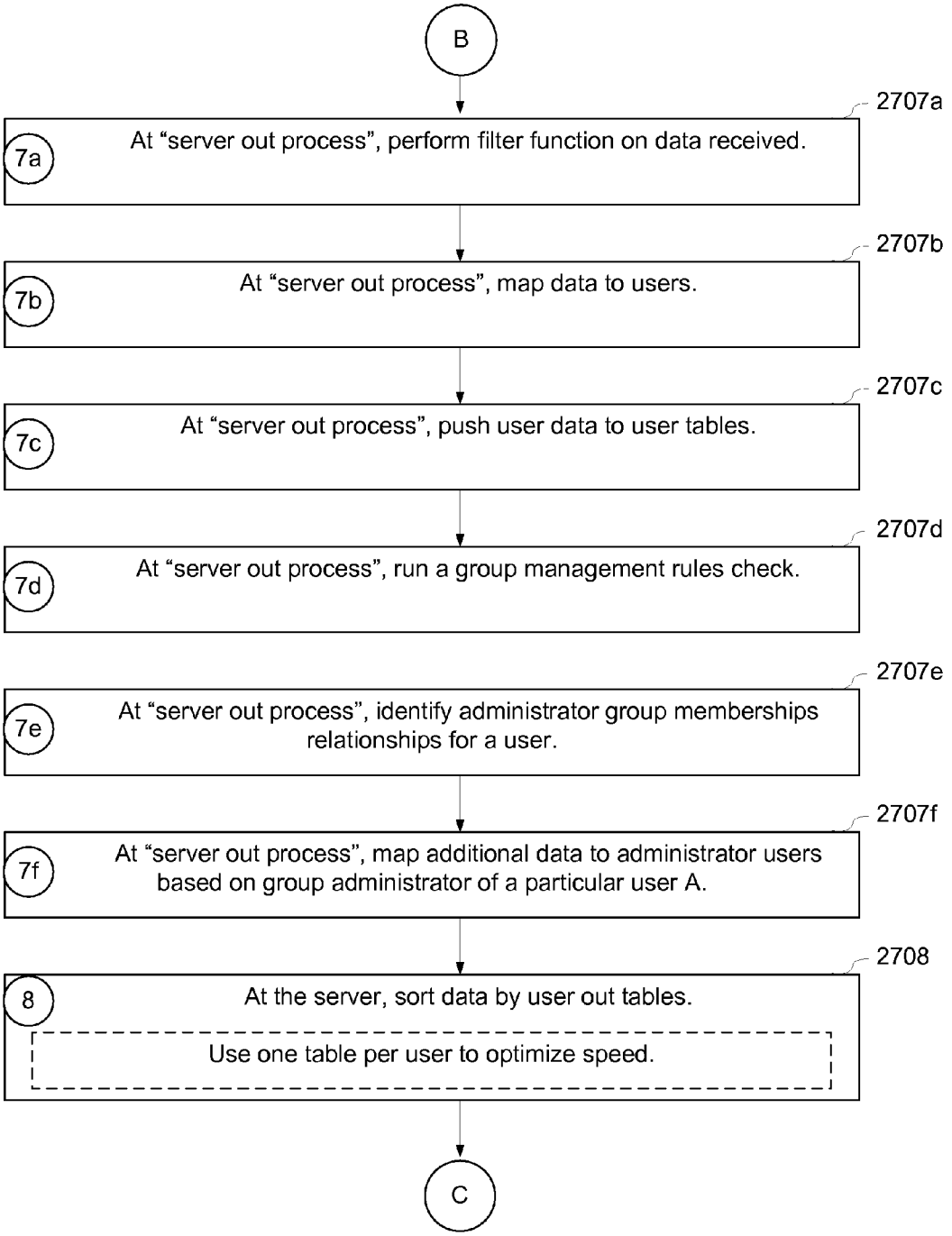

B
2707a
7a
At "server out process", perform filter function on data received.
2707b
7b
At "server out process", map data to users.
2707c
7c
At "server out process", push user data to user tables.
2707d
7d
At "server out process", run a group management rules check.
2707e
7e
At "server out process", identify administrator group memberships relationships for a user.
2707f
7f
At "server out process", map additional data to administrator users based on group administrator of a particular user A.
2708
8
At the server, sort data by user out tables.
Use one table per user to optimize speed.
C

Figure 27B

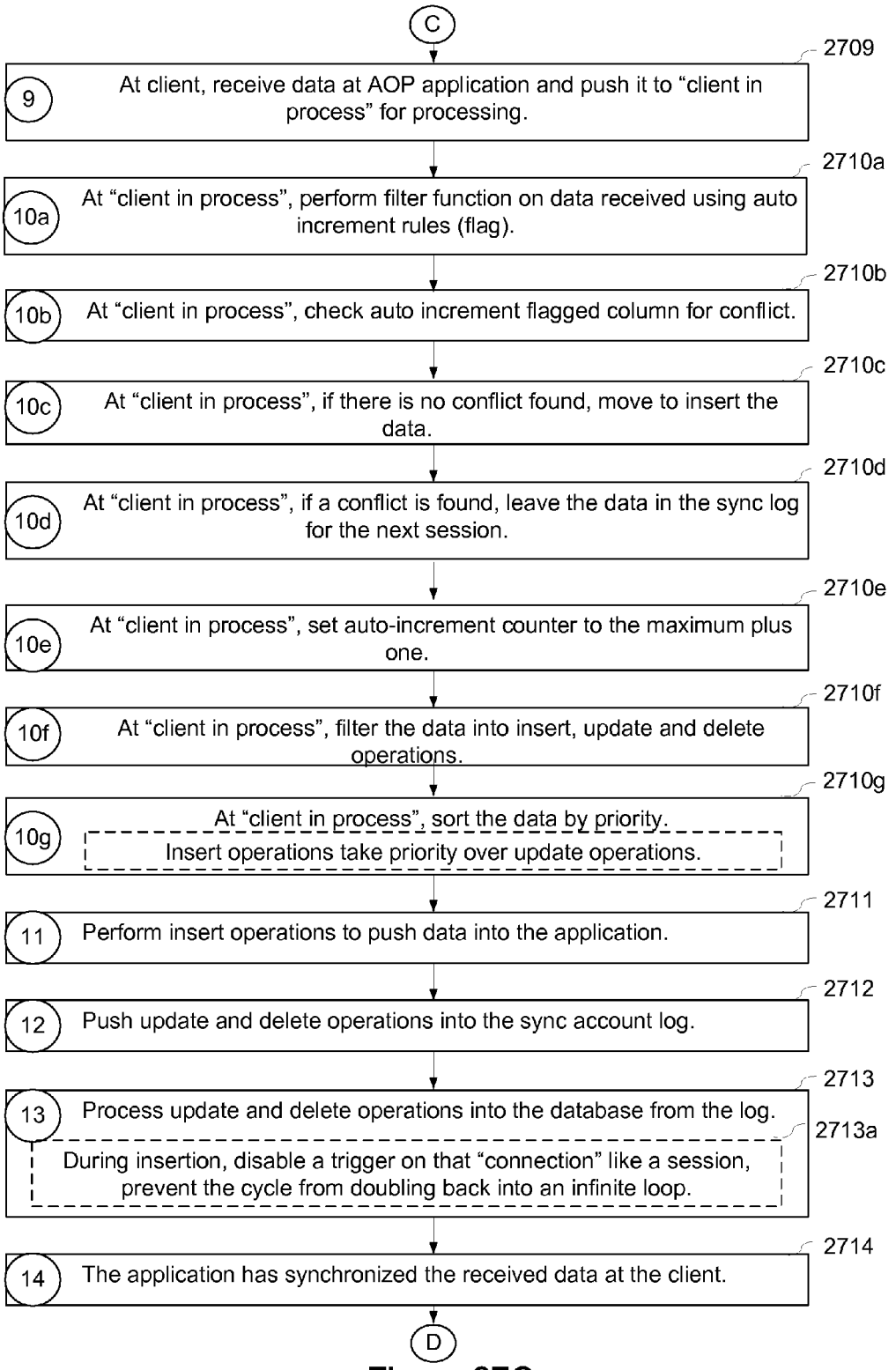

C
2709
9
At client, receive data at AOP application and push it to "client in process" for processing.
2710a
10a
At "client in process", perform filter function on data received using auto increment rules (flag).
2710b
10b
At "client in process", check auto increment flagged column for conflict.
2710c
10c
At "client in process", if there is no conflict found, move to insert the data.
2710d
10d
At "client in process", if a conflict is found, leave the data in the sync log for the next session.
2710e
10e
At "client in process", set auto-increment counter to the maximum plus one.
2710f
10f
At "client in process", filter the data into insert, update and delete operations.
2710g
10g
At "client in process", sort the data by priority.
Insert operations take priority over update operations.
2711
11
Perform insert operations to push data into the application.
2712
12
Push update and delete operations into the sync account log.
2713
13
Process update and delete operations into the database from the log.
2713a
During insertion, disable a trigger on that "connection" like a session, prevent the cycle from doubling back into an infinite loop.
2714
14
The application has synchronized the received data at the client.
D

Figure 27C

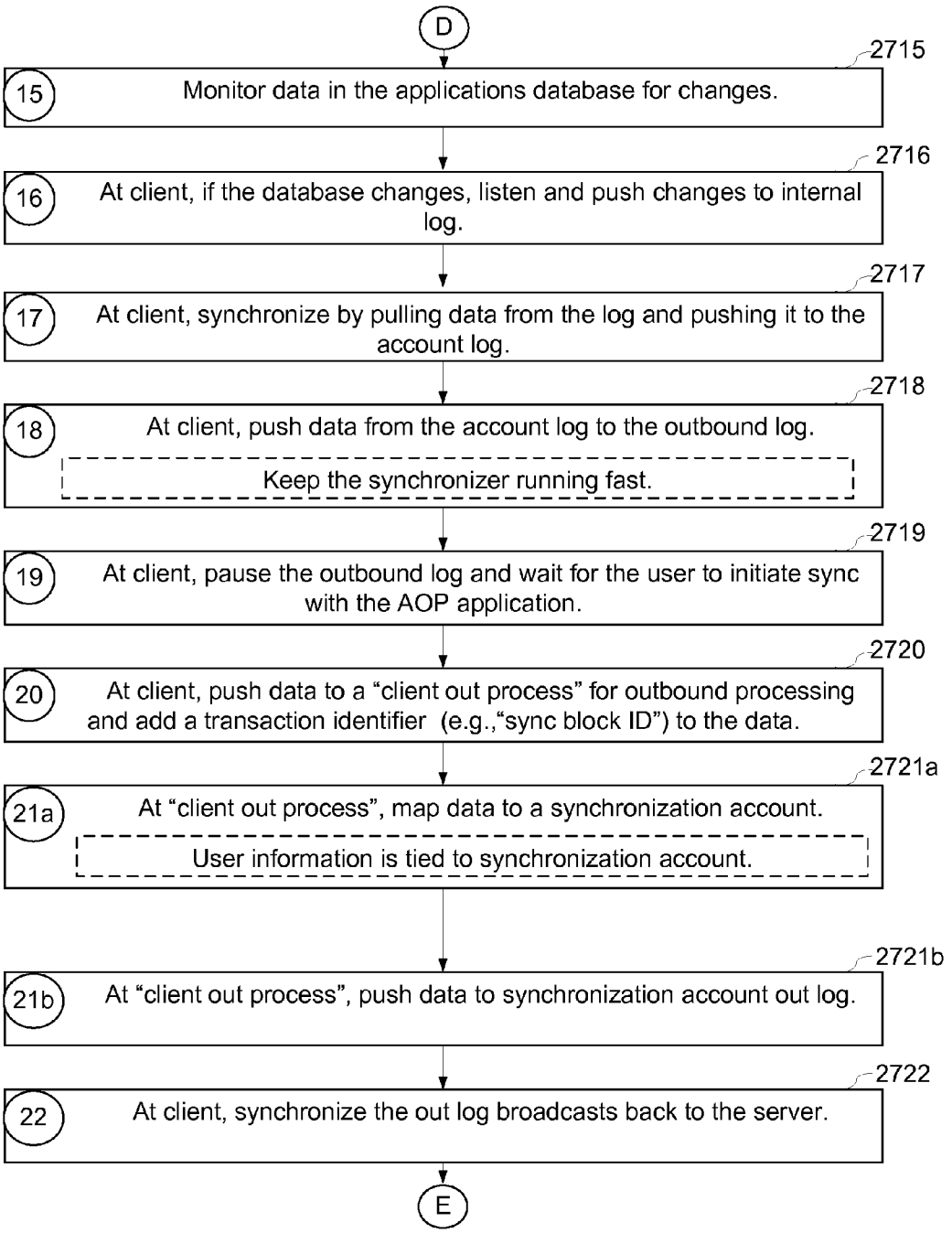

D
2715
15
Monitor data in the applications database for changes.
2716
16
At client, if the database changes, listen and push changes to internal log.
2717
17
At client, synchronize by pulling data from the log and pushing it to the account log.
2718
18
At client, push data from the account log to the outbound log.
Keep the synchronizer running fast.
2719
19
At client, pause the outbound log and wait for the user to initiate sync with the AOP application.
2720
20
At client, push data to a "client out process" for outbound processing and add a transaction identifier (e.g.,"sync block ID") to the data.
2721a
21a
At "client out process", map data to a synchronization account.
User information is tied to synchronization account.
2721b
21b
At "client out process", push data to synchronization account out log.
2722
22
At client, synchronize the out log broadcasts back to the server.
E

Figure 27D

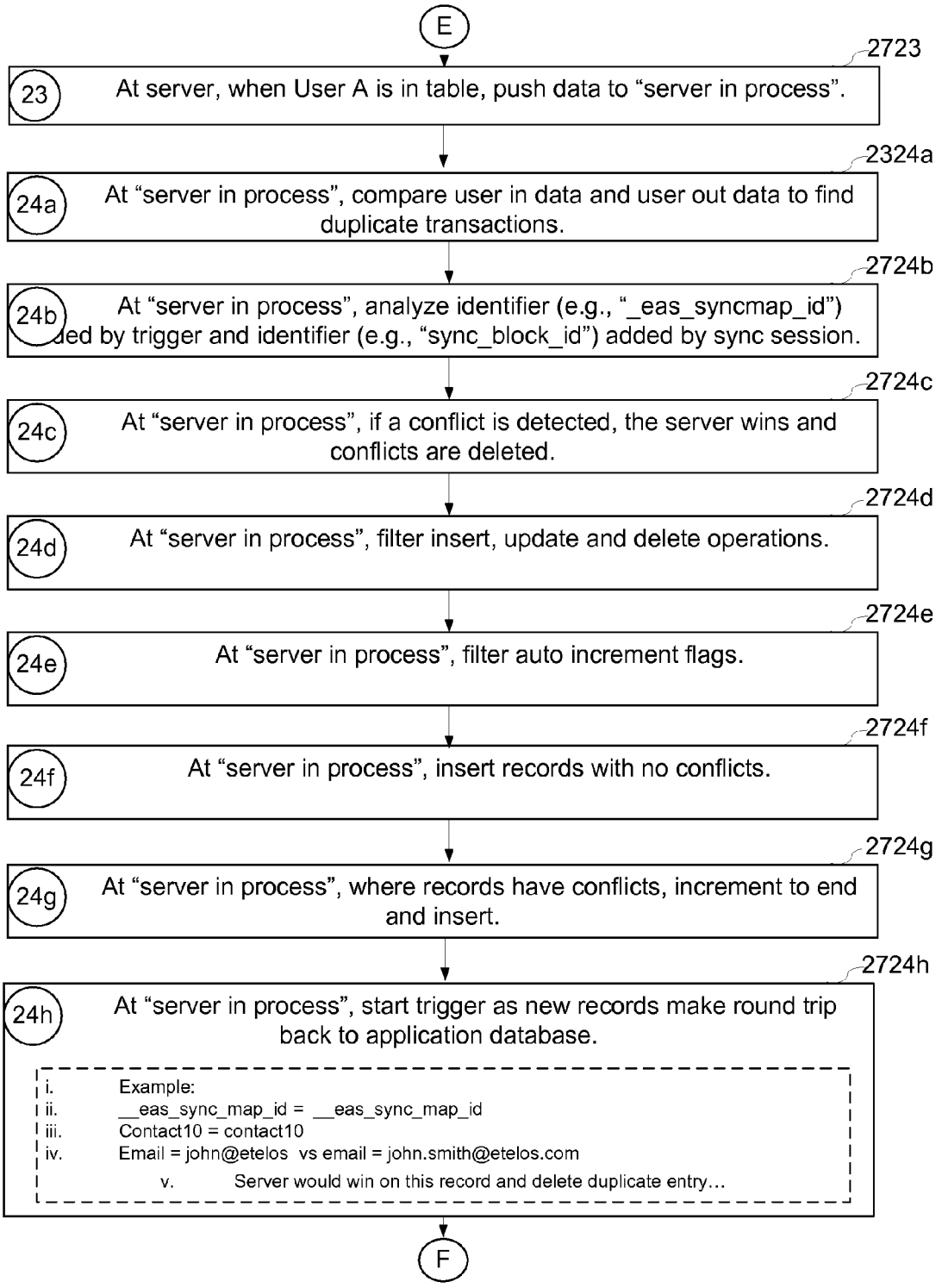

E
2723
23
At server, when User A is in table, push data to "server in process".
2324a
24a
At "server in process", compare user in data and user out data to find duplicate transactions.
2724b
24b
At "server in process", analyze identifier (e.g., "_eas_syncmap_id") ded by trigger and identifier (e.g., "sync_block_id") added by sync session.
2724c
24c
At "server in process", if a conflict is detected, the server wins and conflicts are deleted.
2724d
24d
At "server in process", filter insert, update and delete operations.
2724e
24e
At "server in process", filter auto increment flags.
2724f
24f
At "server in process", insert records with no conflicts.
2724g
24g
At "server in process", where records have conflicts, increment to end and insert.
2724h
24h
At "server in process", start trigger as new records make round trip back to application database.
i. Example:
ii. __eas_sync_map_id = __eas_sync_map_id
iii. Contact10 = contact10
iv. Email = john@etelos vs email = john.smith@etelos.com
v. Server would win on this record and delete duplicate entry…
F

Figure 27E

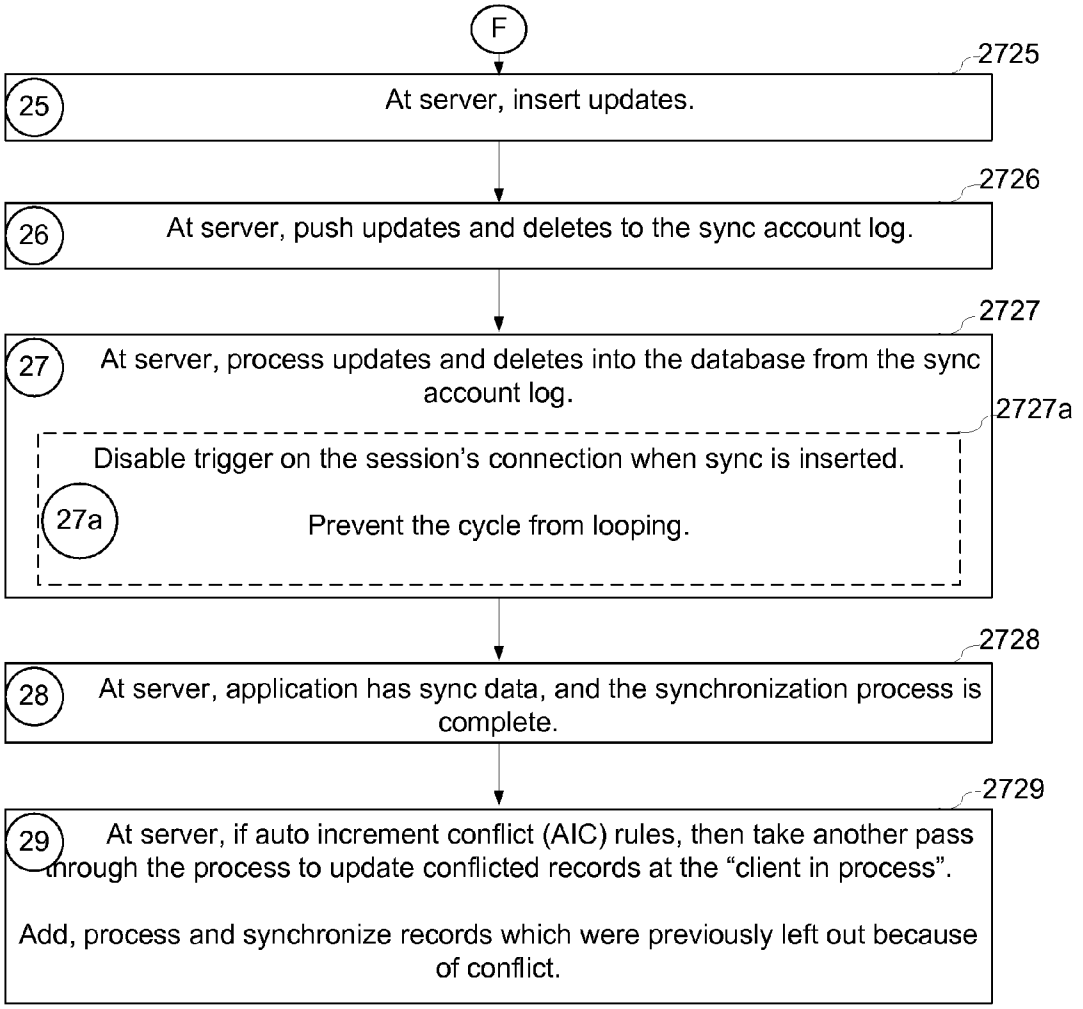

F
2725
25
At server, insert updates.
2726
26
At server, push updates and deletes to the sync account log.
2727
27
At server, process updates and deletes into the database from the sync account log.
2727a
Disable trigger on the session's connection when sync is inserted.
27a
Prevent the cycle from looping.
2728
28
At server, application has sync data, and the synchronization process is complete.
2729
29
At server, if auto increment conflict (AIC) rules, then take another pass through the process to update conflicted records at the "client in process".
Add, process and synchronize records which were previously left out because of conflict.

Figure 27F

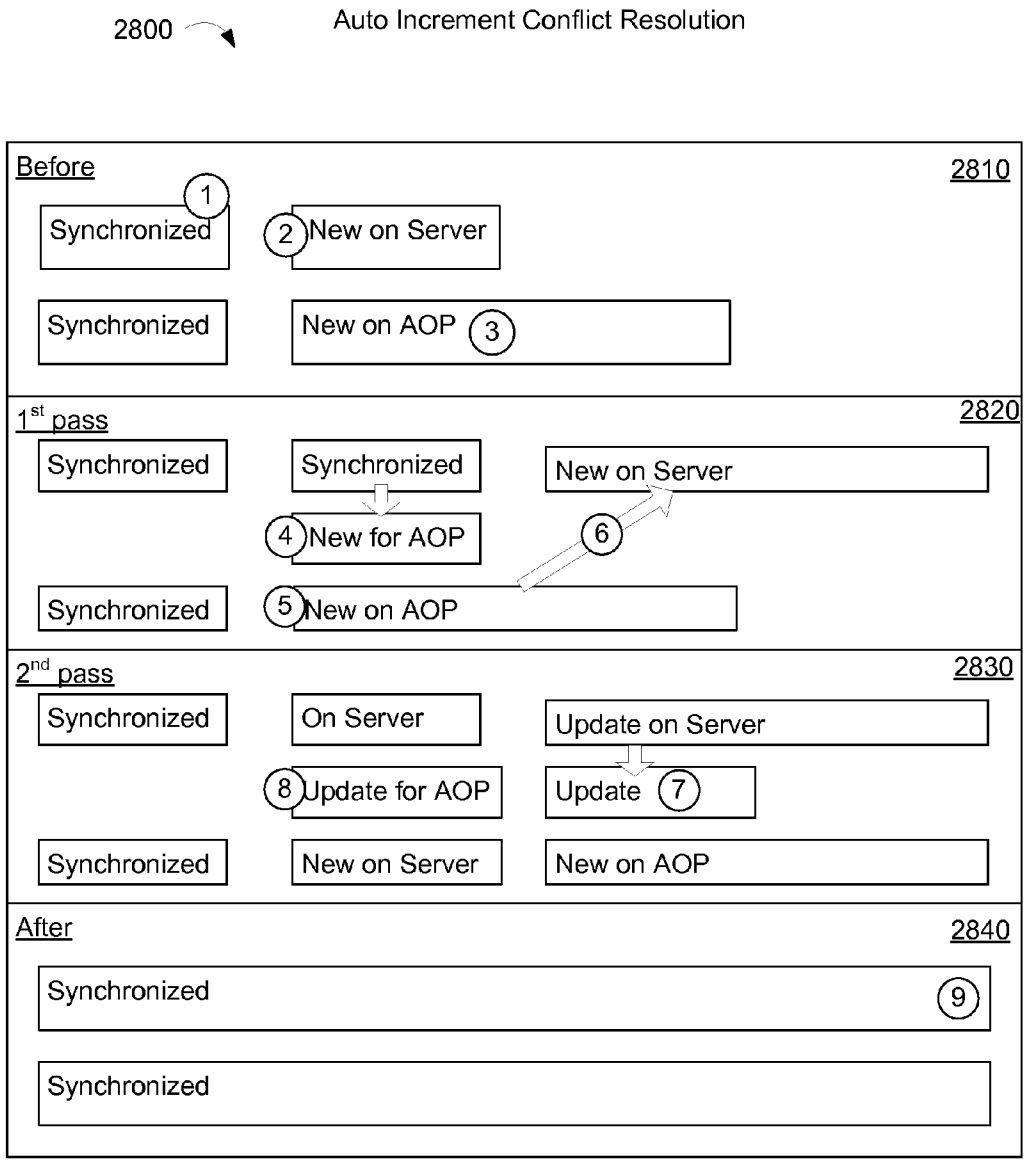

2800
Auto Increment Conflict Resolution
Before
2810
1
Synchronized
2
New on Server
Synchronized
New on AOP
3
1st pass
2820
Synchronized
Synchronized
New on Server
4
New for AOP
6
Synchronized
5
New on AOP
2nd pass
2830
Synchronized
On Server
Update on Server
8
Update for AOP
Update
7
Synchronized
New on Server
New on AOP
After
2840
Synchronized
9
Synchronized

Figure 28

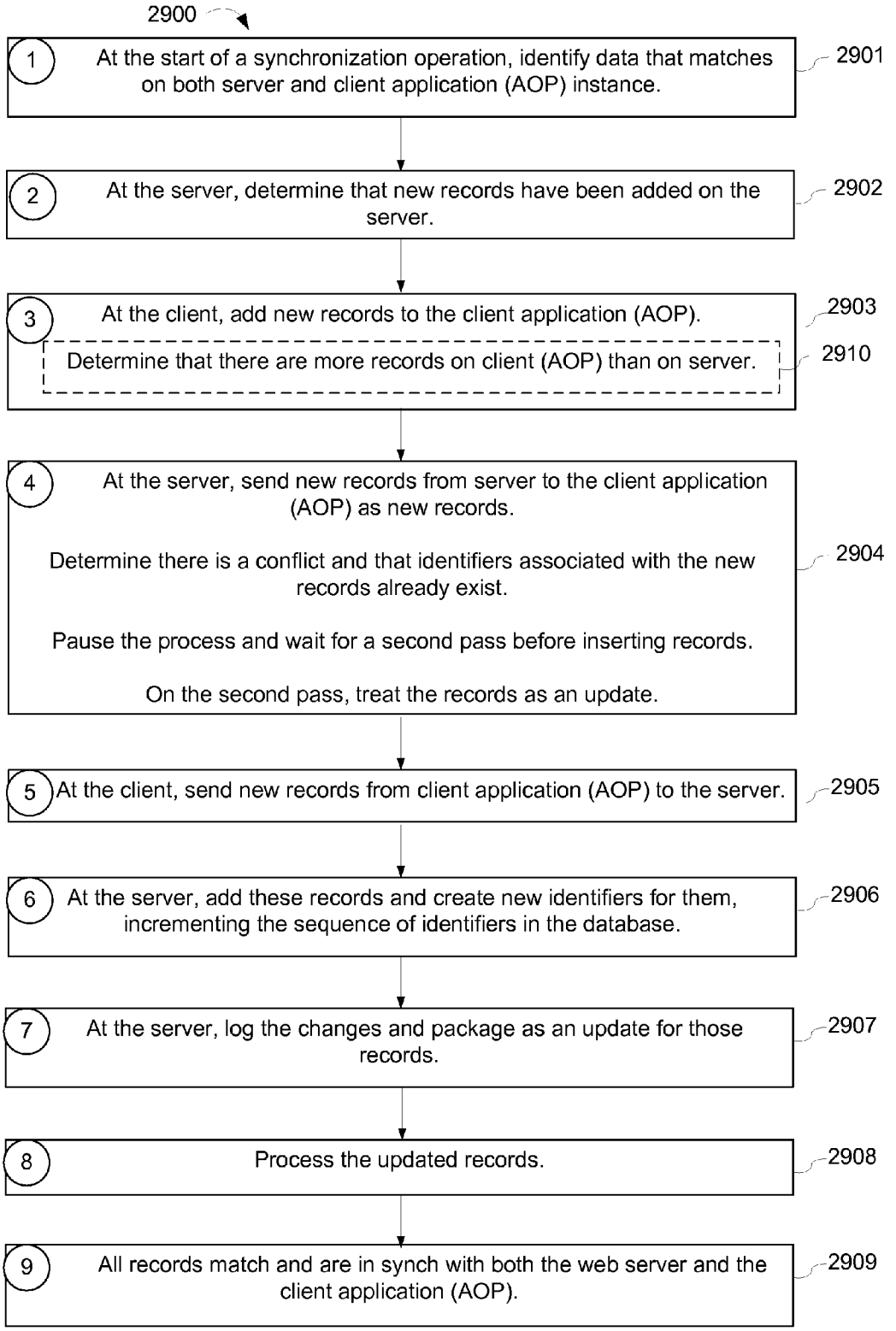

2900
1
At the start of a synchronization operation, identify data that matches on both server and client application (AOP) instance.
2901
2
At the server, determine that new records have been added on the server.
2902
3
At the client, add new records to the client application (AOP).
2903
Determine that there are more records on client (AOP) than on server.
2910
4
At the server, send new records from server to the client application (AOP) as new records.
Determine there is a conflict and that identifiers associated with the new records already exist.
Pause the process and wait for a second pass before inserting records.
On the second pass, treat the records as an update.
2904
5
At the client, send new records from client application (AOP) to the server.
2905
6
At the server, add these records and create new identifiers for them, incrementing the sequence of identifiers in the database.
2906
7
At the server, log the changes and package as an update for those records.
2907
8
Process the updated records.
2908
9
All records match and are in synch with both the web server and the client application (AOP).
2909

Figure 29

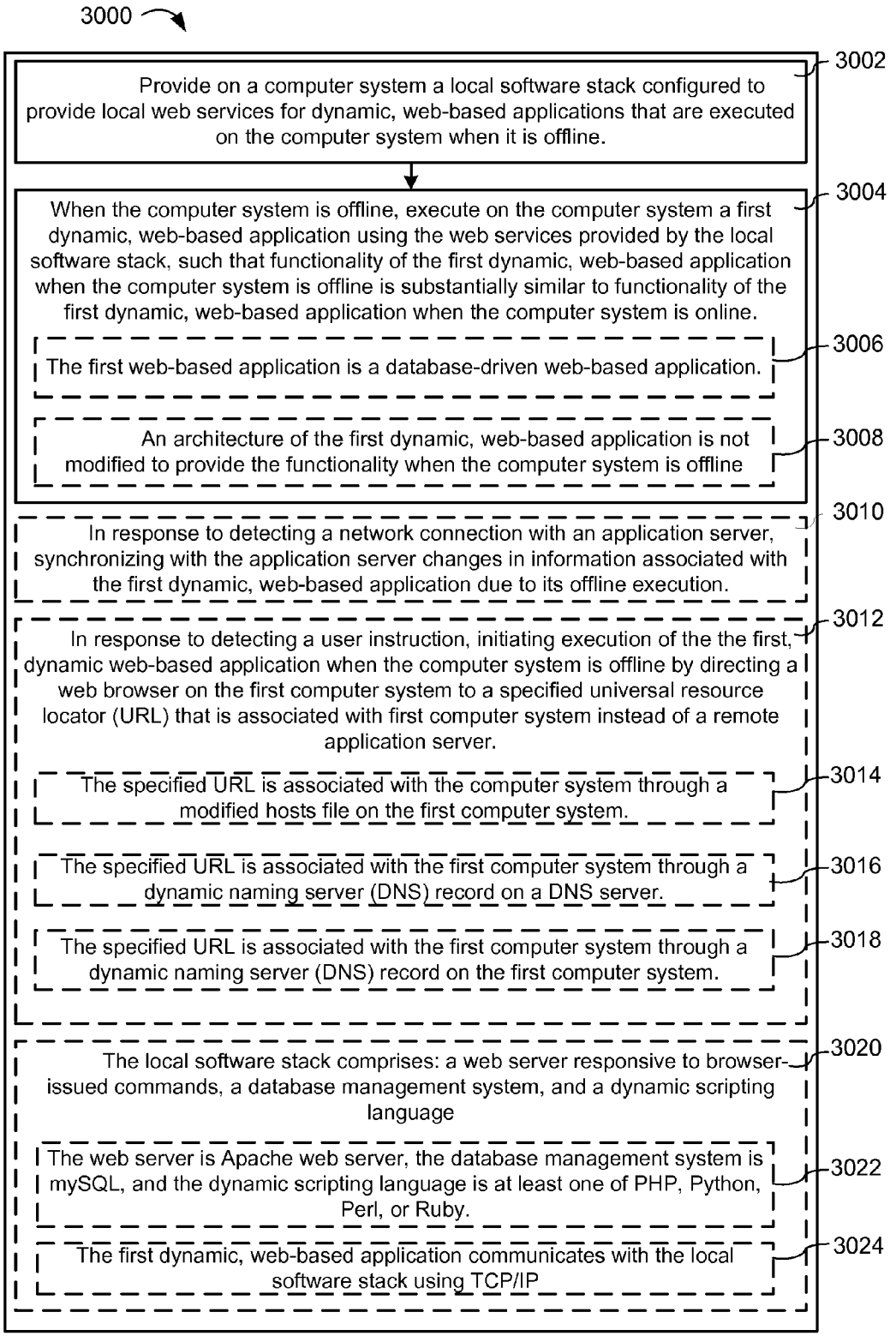

3000
Provide on a computer system a local software stack configured to provide local web services for dynamic, web-based applications that are executed on the computer system when it is offline.
3002
When the computer system is offline, execute on the computer system a first dynamic, web-based application using the web services provided by the local software stack, such that functionality of the first dynamic, web-based application when the computer system is offline is substantially similar to functionality of the first dynamic, web-based application when the computer system is online.
3004
The first web-based application is a database-driven web-based application.
3006
An architecture of the first dynamic, web-based application is not modified to provide the functionality when the computer system is offline
3008
In response to detecting a network connection with an application server, synchronizing with the application server changes in information associated with the first dynamic, web-based application due to its offline execution.
3010
In response to detecting a user instruction, initiating execution of the the first, dynamic web-based application when the computer system is offline by directing a web browser on the first computer system to a specified universal resource locator (URL) that is associated with first computer system instead of a remote application server.
3012
The specified URL is associated with the computer system through a modified hosts file on the first computer system.
3014
The specified URL is associated with the first computer system through a dynamic naming server (DNS) record on a DNS server.
3016
The specified URL is associated with the first computer system through a dynamic naming server (DNS) record on the first computer system.
3018
The local software stack comprises: a web server responsive to browser-issued commands, a database management system, and a dynamic scripting language
3020
The web server is Apache web server, the database management system is mySQL, and the dynamic scripting language is at least one of PHP, Python, Perl, or Ruby.
3022
The first dynamic, web-based application communicates with the local software stack using TCP/IP
3024

Figure 30

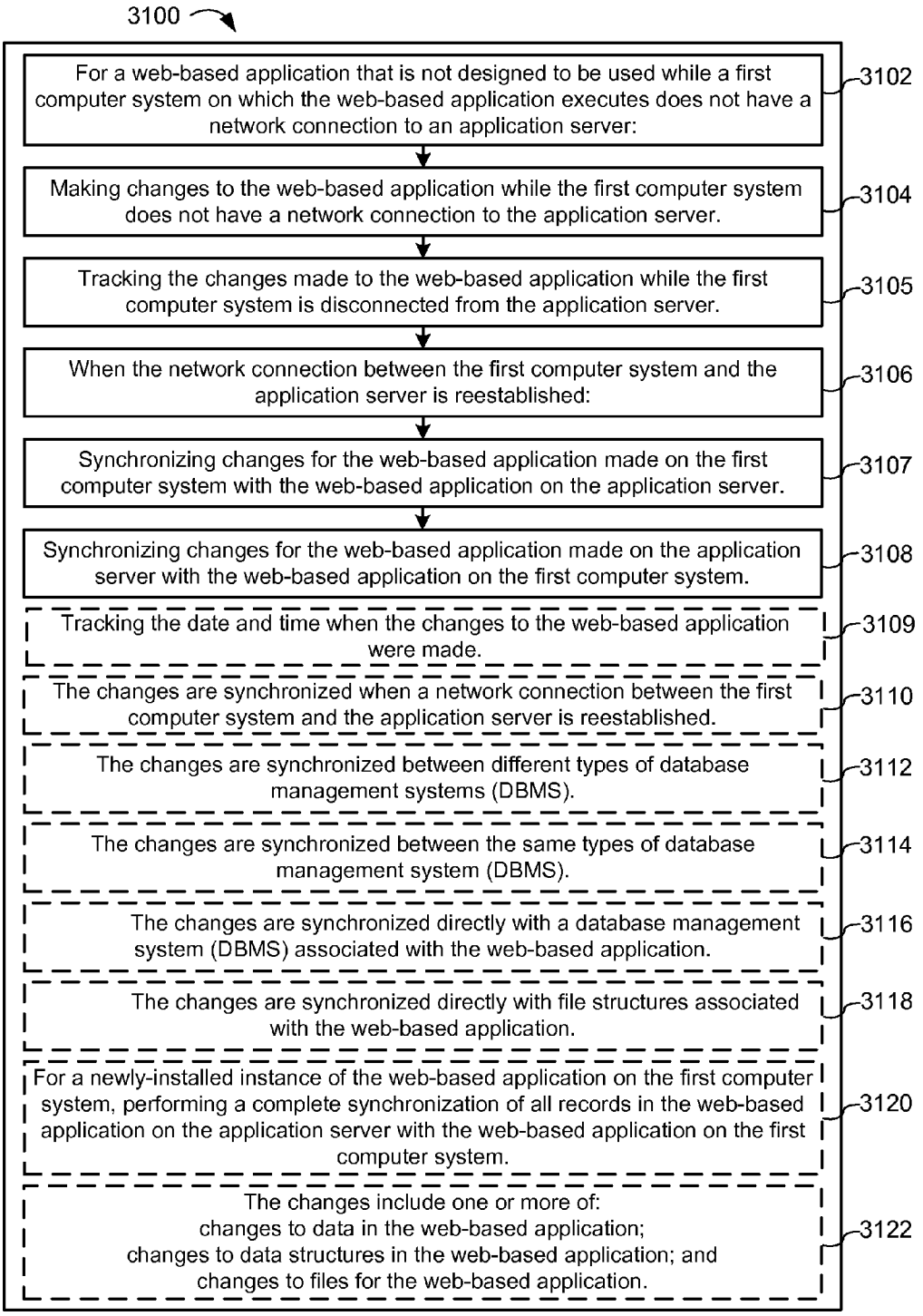

3100
For a web-based application that is not designed to be used while a first computer system on which the web-based application executes does not have a network connection to an application server:
3102
Making changes to the web-based application while the first computer system does not have a network connection to the application server.
3104
Tracking the changes made to the web-based application while the first computer system is disconnected from the application server.
3105
When the network connection between the first computer system and the application server is reestablished:
3106
Synchronizing changes for the web-based application made on the first computer system with the web-based application on the application server.
3107
Synchronizing changes for the web-based application made on the application server with the web-based application on the first computer system.
3108
Tracking the date and time when the changes to the web-based application were made.
3109
The changes are synchronized when a network connection between the first computer system and the application server is reestablished.
3110
The changes are synchronized between different types of database management systems (DBMS).
3112
The changes are synchronized between the same types of database management system (DBMS).
3114
The changes are synchronized directly with a database management system (DBMS) associated with the web-based application.
3116
The changes are synchronized directly with file structures associated with the web-based application.
3118
For a newly-installed instance of the web-based application on the first computer system, performing a complete synchronization of all records in the web-based application on the application server with the web-based application on the first computer system.
3120
The changes include one or more of:
changes to data in the web-based application;
changes to data structures in the web-based application; and
changes to files for the web-based application.
3122

Figure 31

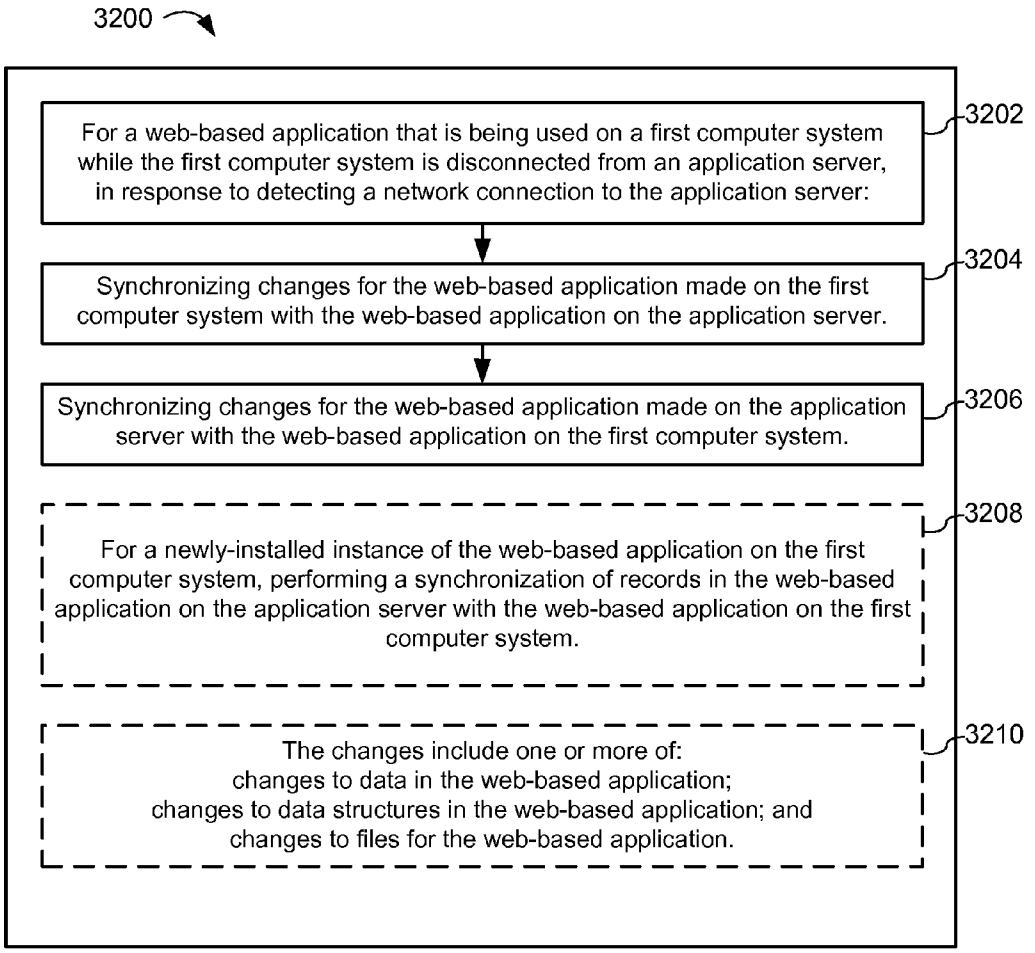

3200
For a web-based application that is being used on a first computer system while the first computer system is disconnected from an application server, in response to detecting a network connection to the application server:
3202
Synchronizing changes for the web-based application made on the first computer system with the web-based application on the application server.
3204
Synchronizing changes for the web-based application made on the application server with the web-based application on the first computer system.
3206
For a newly-installed instance of the web-based application on the first computer system, performing a synchronization of records in the web-based application on the application server with the web-based application on the first computer system.
3208
The changes include one or more of:
changes to data in the web-based application;
changes to data structures in the web-based application; and
changes to files for the web-based application.
3210

Figure 32

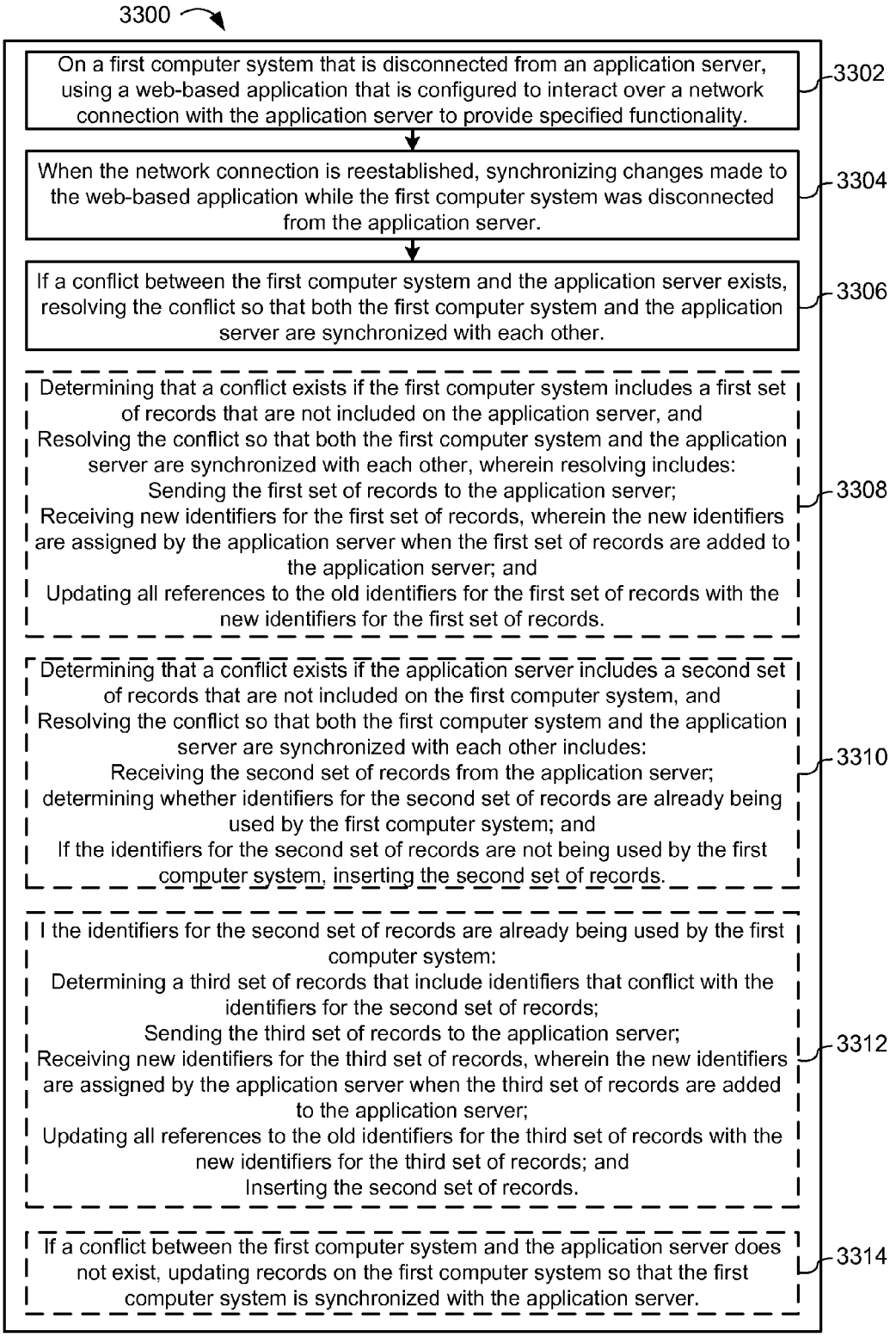

3300
On a first computer system that is disconnected from an application server, using a web-based application that is configured to interact over a network connection with the application server to provide specified functionality.
3302
When the network connection is reestablished, synchronizing changes made to the web-based application while the first computer system was disconnected from the application server.
3304
If a conflict between the first computer system and the application server exists, resolving the conflict so that both the first computer system and the application server are synchronized with each other.
3306
Determining that a conflict exists if the first computer system includes a first set of records that are not included on the application server, and
Resolving the conflict so that both the first computer system and the application server are synchronized with each other, wherein resolving includes:
Sending the first set of records to the application server;
Receiving new identifiers for the first set of records, wherein the new identifiers are assigned by the application server when the first set of records are added to the application server; and
Updating all references to the old identifiers for the first set of records with the new identifiers for the first set of records.
3308
Determining that a conflict exists if the application server includes a second set of records that are not included on the first computer system, and
Resolving the conflict so that both the first computer system and the application server are synchronized with each other includes:
Receiving the second set of records from the application server;
determining whether identifiers for the second set of records are already being used by the first computer system; and
If the identifiers for the second set of records are not being used by the first computer system, inserting the second set of records.
3310
I the identifiers for the second set of records are already being used by the first computer system:
Determining a third set of records that include identifiers that conflict with the identifiers for the second set of records;
Sending the third set of records to the application server;
Receiving new identifiers for the third set of records, wherein the new identifiers are assigned by the application server when the third set of records are added to the application server;
Updating all references to the old identifiers for the third set of records with the new identifiers for the third set of records; and
Inserting the second set of records.
3312
If a conflict between the first computer system and the application server does not exist, updating records on the first computer system so that the first computer system is synchronized with the application server.
3314

Figure 33

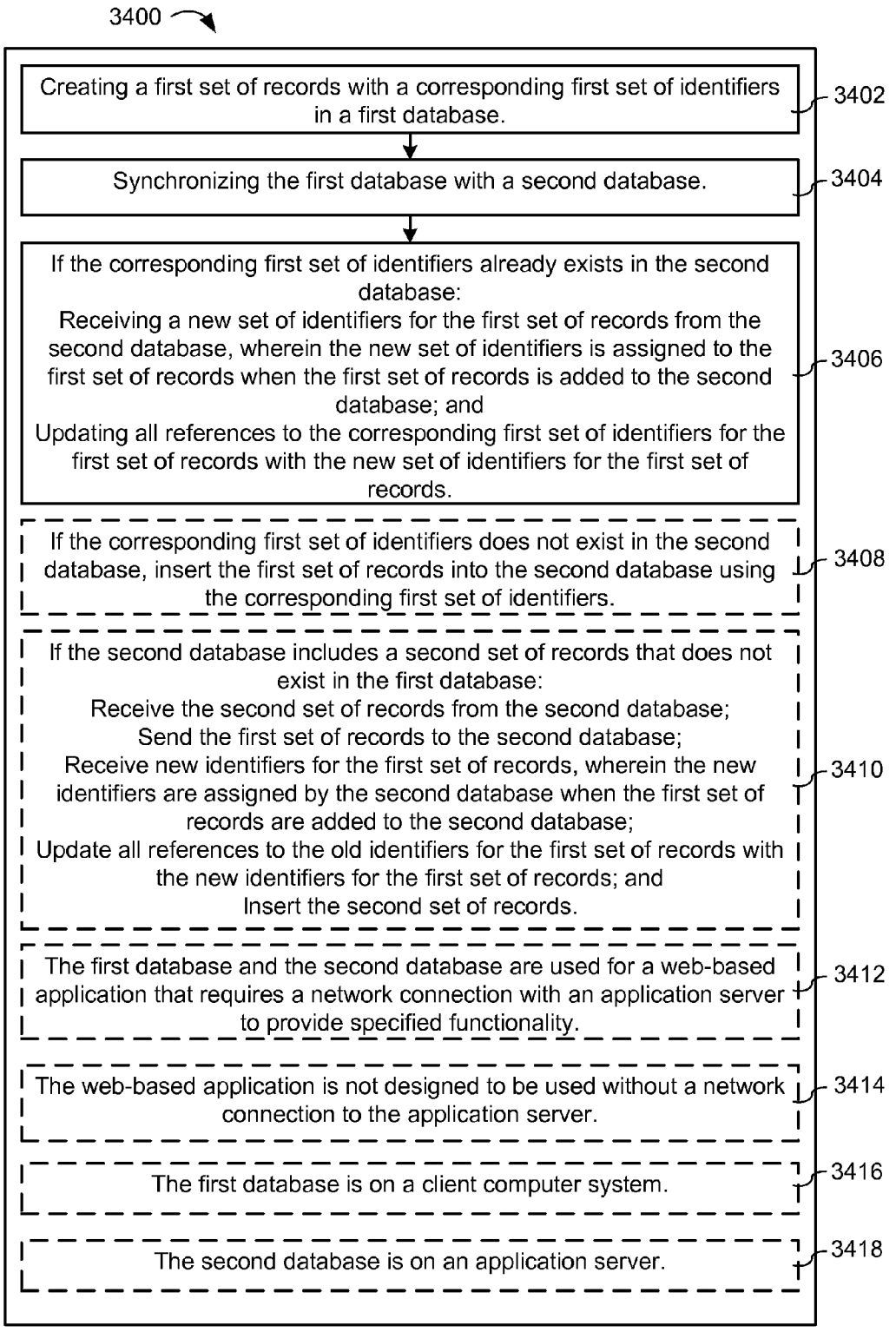

3400
Creating a first set of records with a corresponding first set of identifiers in a first database.
3402
Synchronizing the first database with a second database.
3404
If the corresponding first set of identifiers already exists in the second database:
Receiving a new set of identifiers for the first set of records from the second database, wherein the new set of identifiers is assigned to the first set of records when the first set of records is added to the second database; and
Updating all references to the corresponding first set of identifiers for the first set of records with the new set of identifiers for the first set of records.
3406
If the corresponding first set of identifiers does not exist in the second database, insert the first set of records into the second database using the corresponding first set of identifiers.
3408
If the second database includes a second set of records that does not exist in the first database:
Receive the second set of records from the second database;
Send the first set of records to the second database;
Receive new identifiers for the first set of records, wherein the new identifiers are assigned by the second database when the first set of records are added to the second database;
Update all references to the old identifiers for the first set of records with the new identifiers for the first set of records; and
Insert the second set of records.
3410
The first database and the second database are used for a web-based application that requires a network connection with an application server to provide specified functionality.
3412
The web-based application is not designed to be used without a network connection to the application server.
3414
The first database is on a client computer system.
3416
The second database is on an application server.
3418

Figure 34

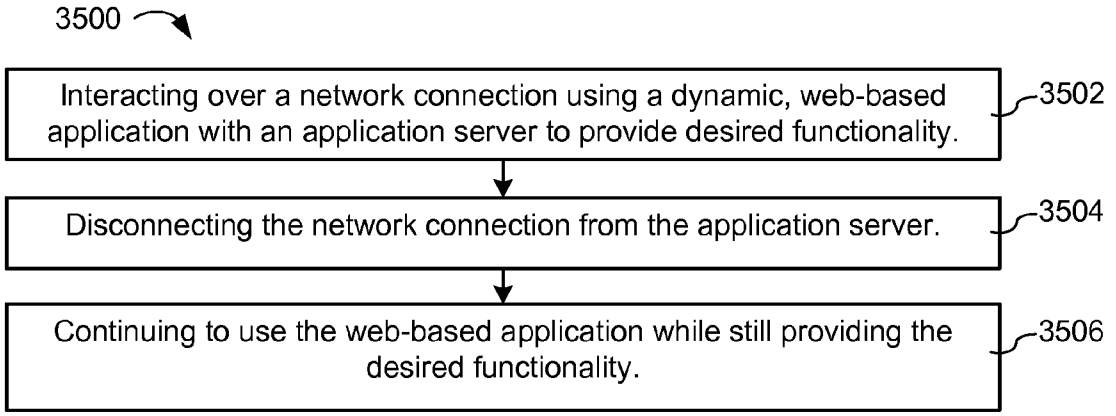

3500
Interacting over a network connection using a dynamic, web-based application with an application server to provide desired functionality.
3502
Disconnecting the network connection from the application server.
3504
Continuing to use the web-based application while still providing the desired functionality.
3506

Figure 35

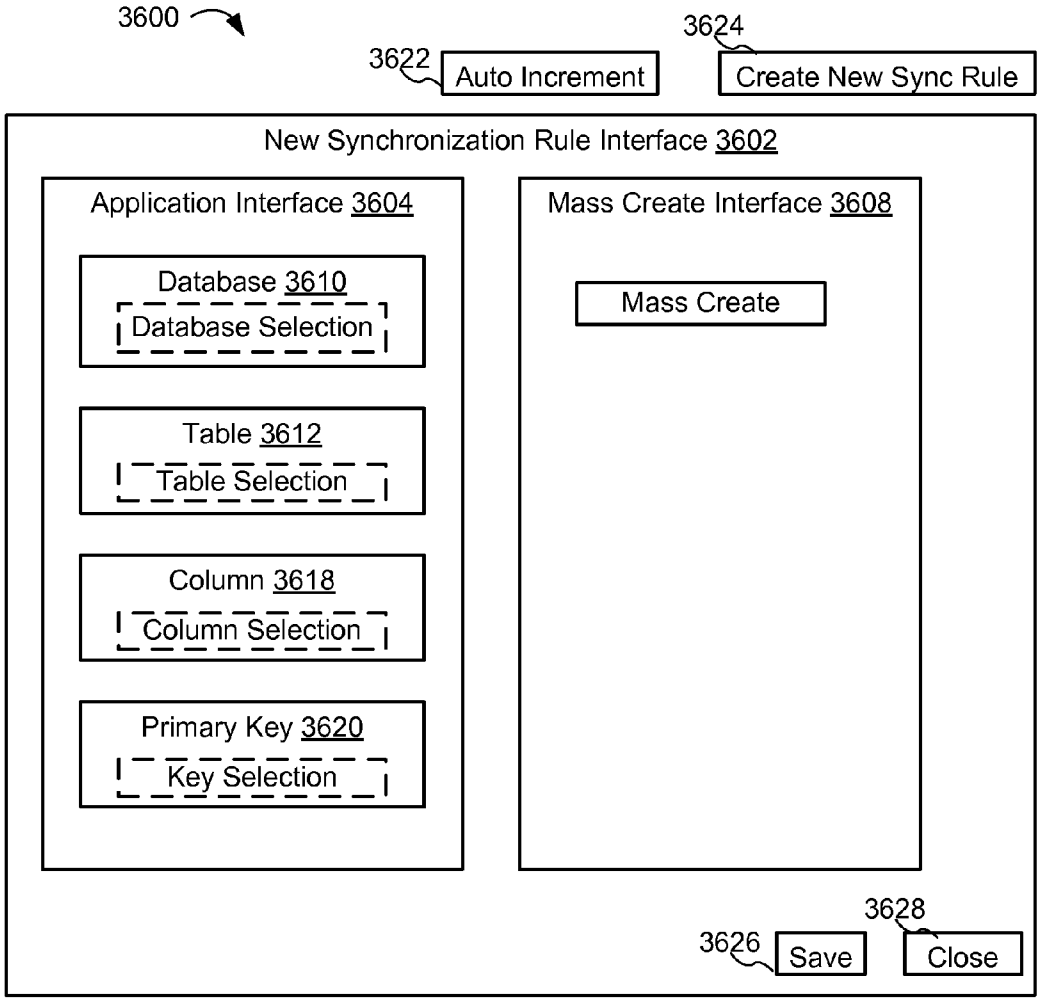

3600
3622
Auto Increment
3624
Create New Sync Rule
New Synchronization Rule Interface 3602
Application Interface 3604
Database 3610
Database Selection
Table 3612
Table Selection
Column 3618
Column Selection
Primary Key 3620
Key Selection
Mass Create Interface 3608
Mass Create
3626
Save
3628
Close

Figure 36

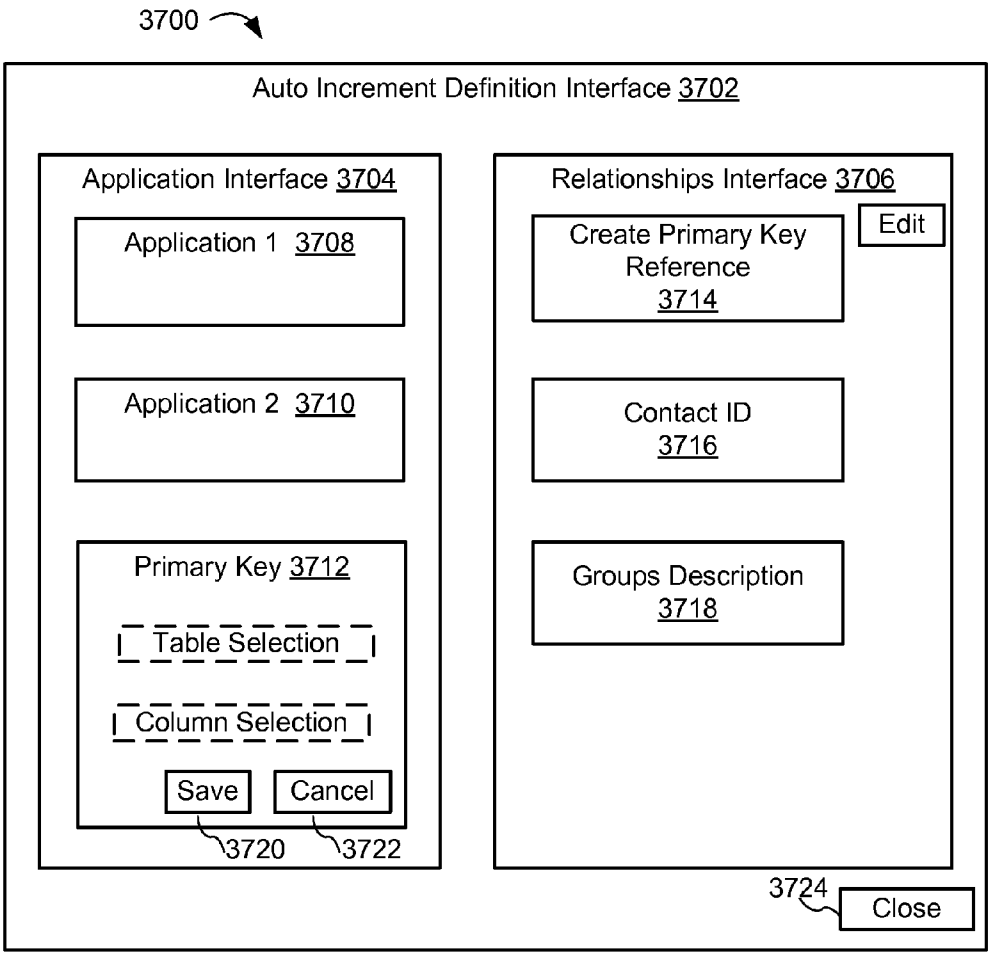

3700
Auto Increment Definition Interface 3702
Application Interface 3704
Application 1 3708
Application 2 3710
Primary Key 3712
Table Selection
Column Selection
Save
Cancel
3720
3722
Relationships Interface 3706
Edit
Create Primary Key Reference 3714
Contact ID 3716
Groups Description 3718
3724
Close

Figure 37

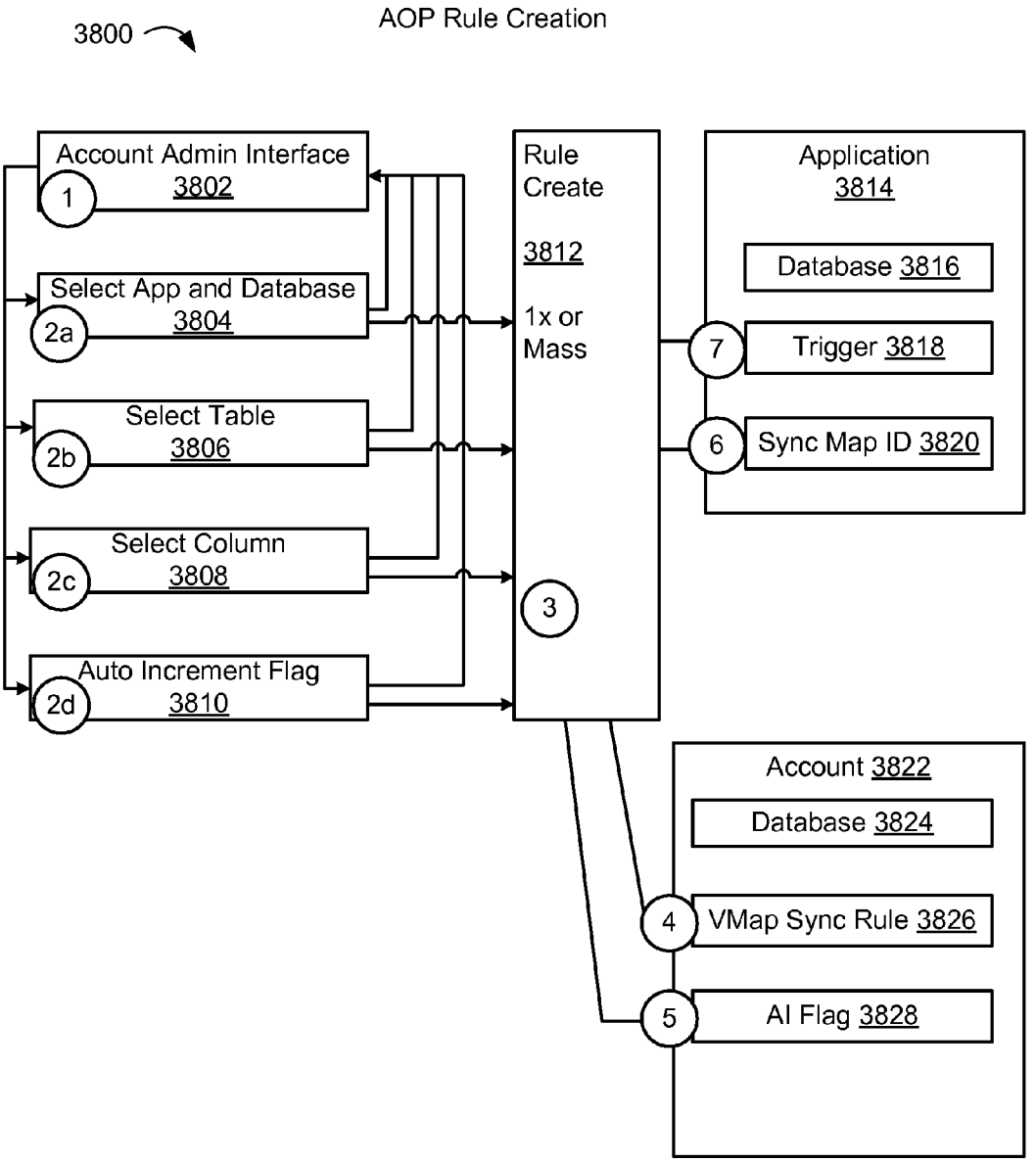

3800
AOP Rule Creation
Account Admin Interface
3802
1
Select App and Database
3804
2a
Select Table
3806
2b
Select Column
3808
2c
Auto Increment Flag
3810
2d
Rule
Create
3812
1x or
Mass
3
Application
3814
Database 3816
Trigger 3818
7
Sync Map ID 3820
6
Account 3822
Database 3824
4
VMap Sync Rule 3826
5
AI Flag 3828

Figure 38

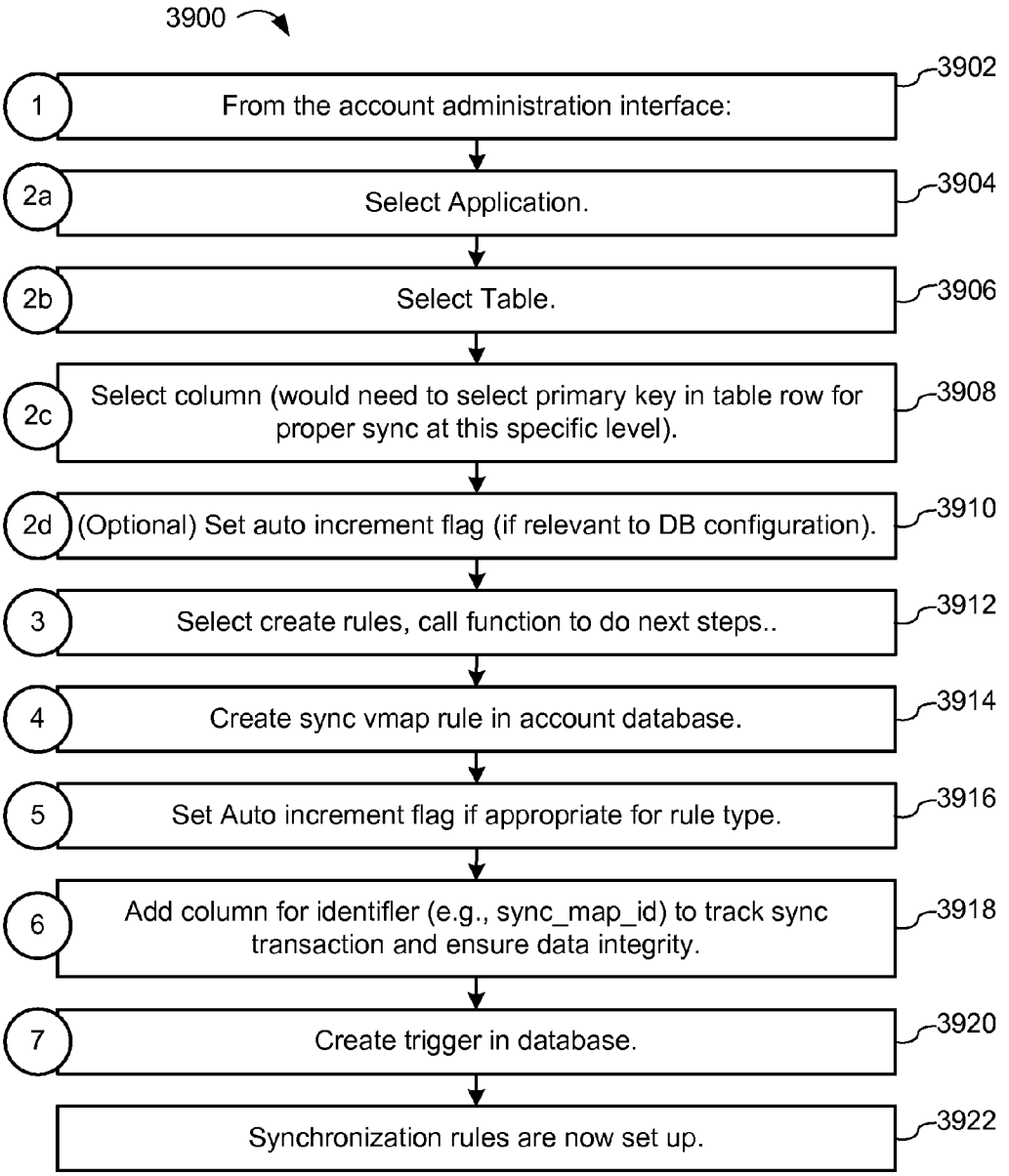

3900
1
From the account administration interface:
3902
2a
Select Application.
3904
2b
Select Table.
3906
2c
Select column (would need to select primary key in table row for proper sync at this specific level).
3908
2d
(Optional) Set auto increment flag (if relevant to DB configuration).
3910
3
Select create rules, call function to do next steps..
3912
4
Create sync vmap rule in account database.
3914
5
Set Auto increment flag if appropriate for rule type.
3916
6
Add column for identifler (e.g., sync_map_id) to track sync transaction and ensure data integrity.
3918
7
Create trigger in database.
3920
Synchronization rules are now set up.
3922

Figure 39

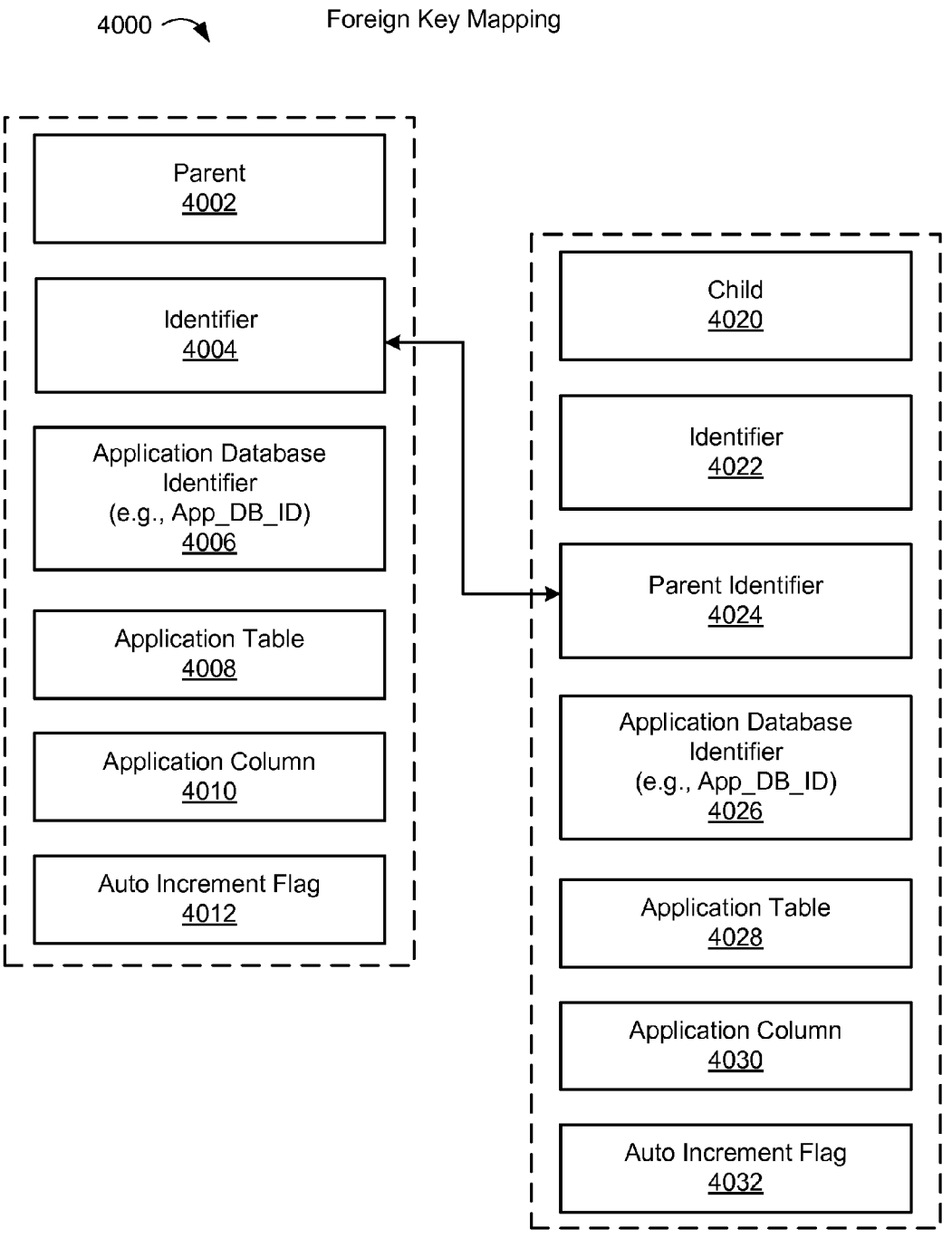

4000
Foreign Key Mapping
Parent
4002
Identifier
4004
Application Database
Identifier
(e.g., App_DB_ID)
4006
Application Table
4008
Application Column
4010
Auto Increment Flag
4012
Child
4020
Identifier
4022
Parent Identifier
4024
Application Database
Identifier
(e.g., App_DB_ID)
4026
Application Table
4028
Application Column
4030
Auto Increment Flag
4032

Figure 40

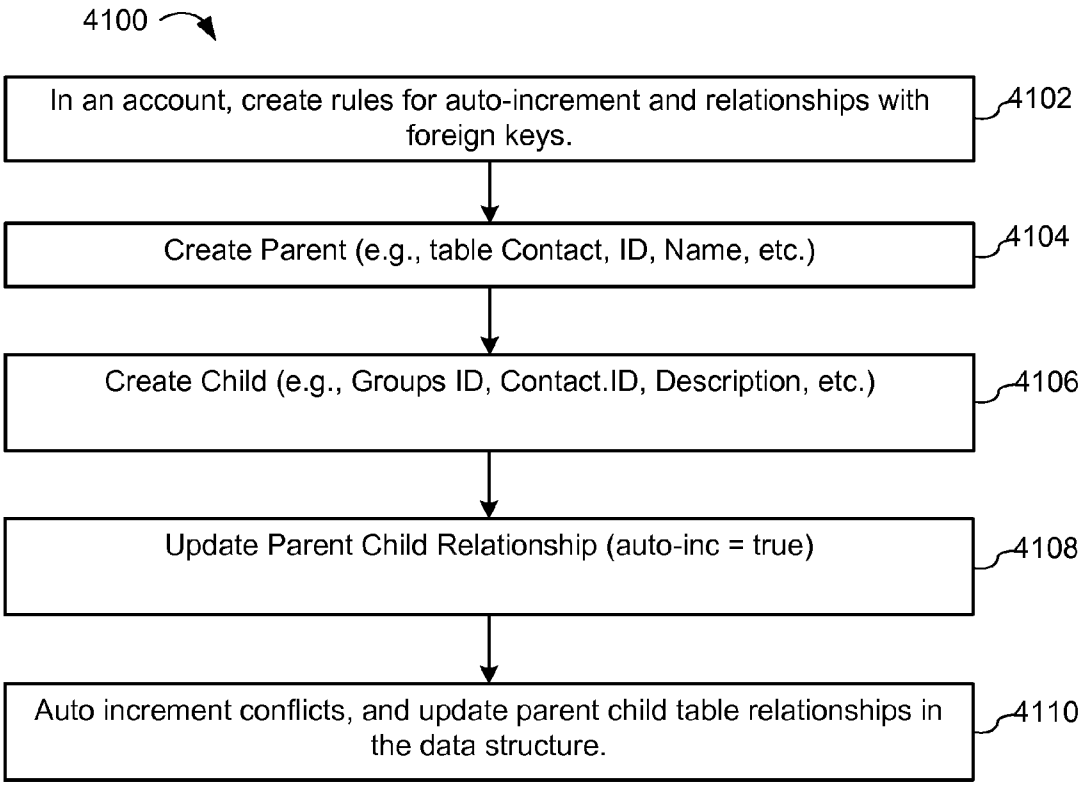

4100
In an account, create rules for auto-increment and relationships with foreign keys.
4102
Create Parent (e.g., table Contact, ID, Name, etc.)
4104
Create Child (e.g., Groups ID, Contact.ID, Description, etc.)
4106
Update Parent Child Relationship (auto-inc = true)
4108
Auto increment conflicts, and update parent child table relationships in the data structure.
4110

Figure 41

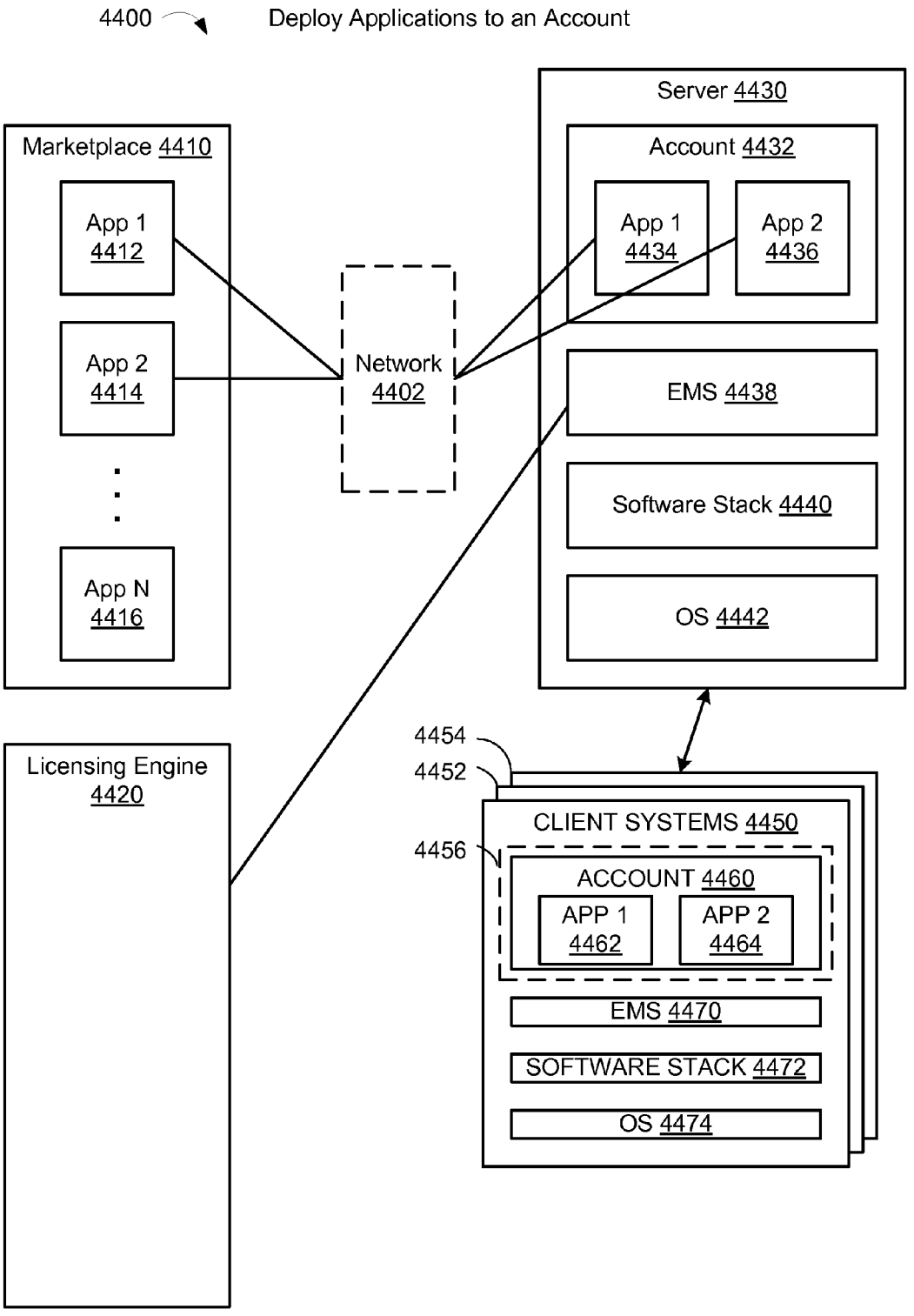

4400
Deploy Applications to an Account
Server 4430
Marketplace 4410
Account 4432
App 1 4412
App 2 4414
App N 4416
App 1 4434
App 2 4436
Network 4402
EMS 4438
Software Stack 4440
OS 4442
Licensing Engine 4420
4454
4452
CLIENT SYSTEMS 4450
4456
ACCOUNT 4460
APP 1 4462
APP 2 4464
EMS 4470
SOFTWARE STACK 4472
OS 4474

Figure 44

Multitenancy License Management

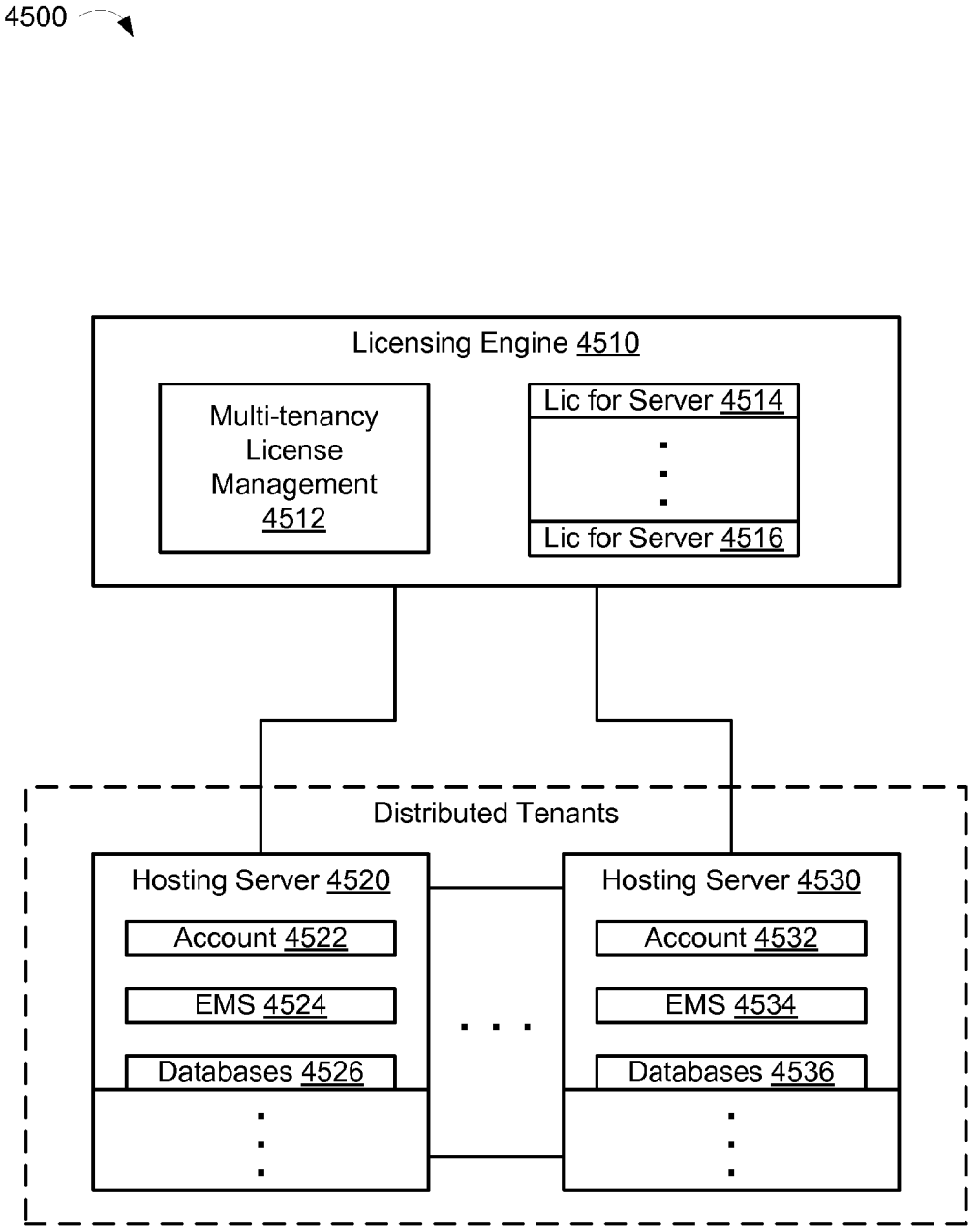


Figure 45

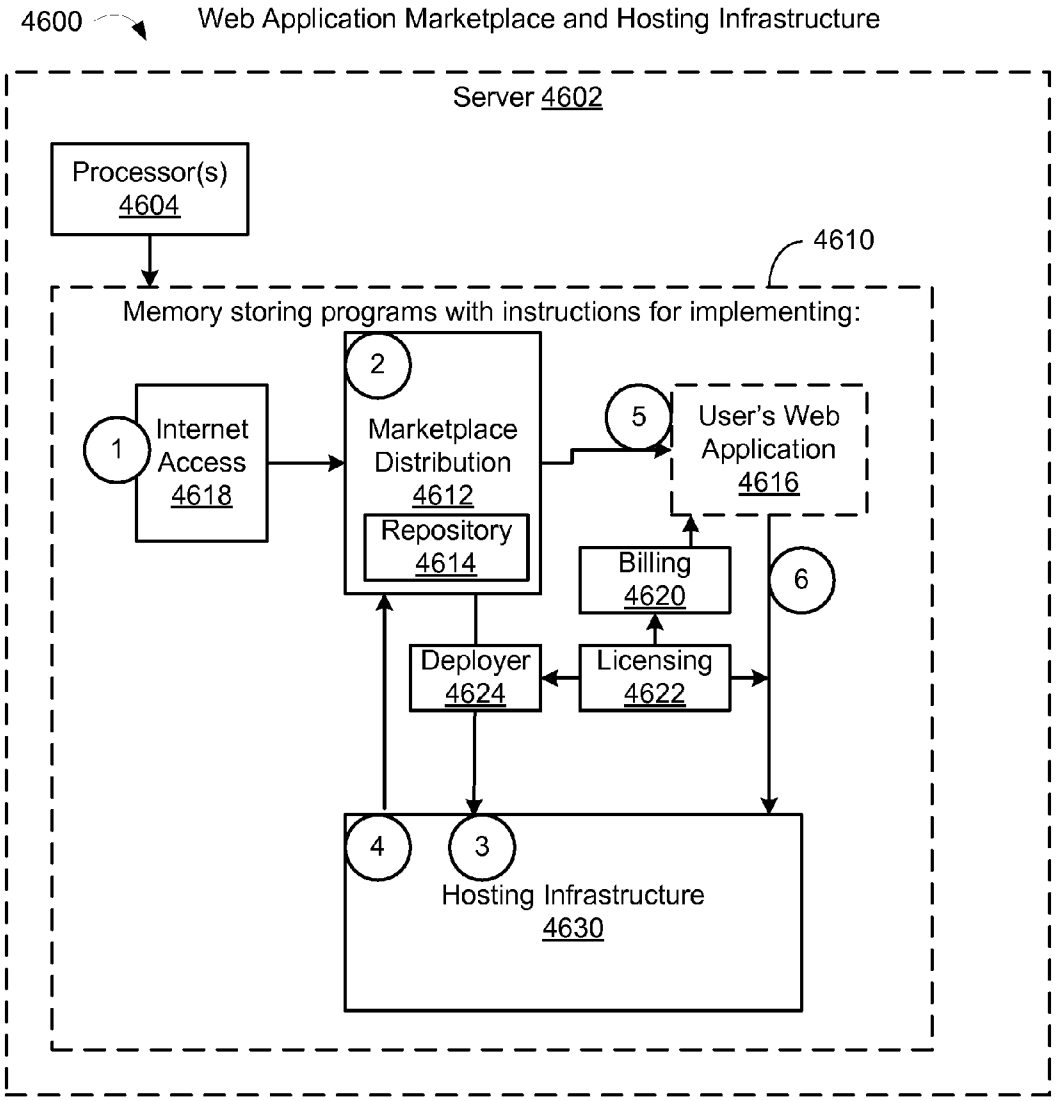

4600
Web Application Marketplace and Hosting Infrastructure
Server 4602
Processor(s) 4604
4610
Memory storing programs with instructions for implementing:
1
Internet Access 4618
2
Marketplace Distribution 4612
Repository 4614
5
User's Web Application 4616
Billing 4620
6
Deployer 4624
Licensing 4622
4
3
Hosting Infrastructure 4630

Figure 46

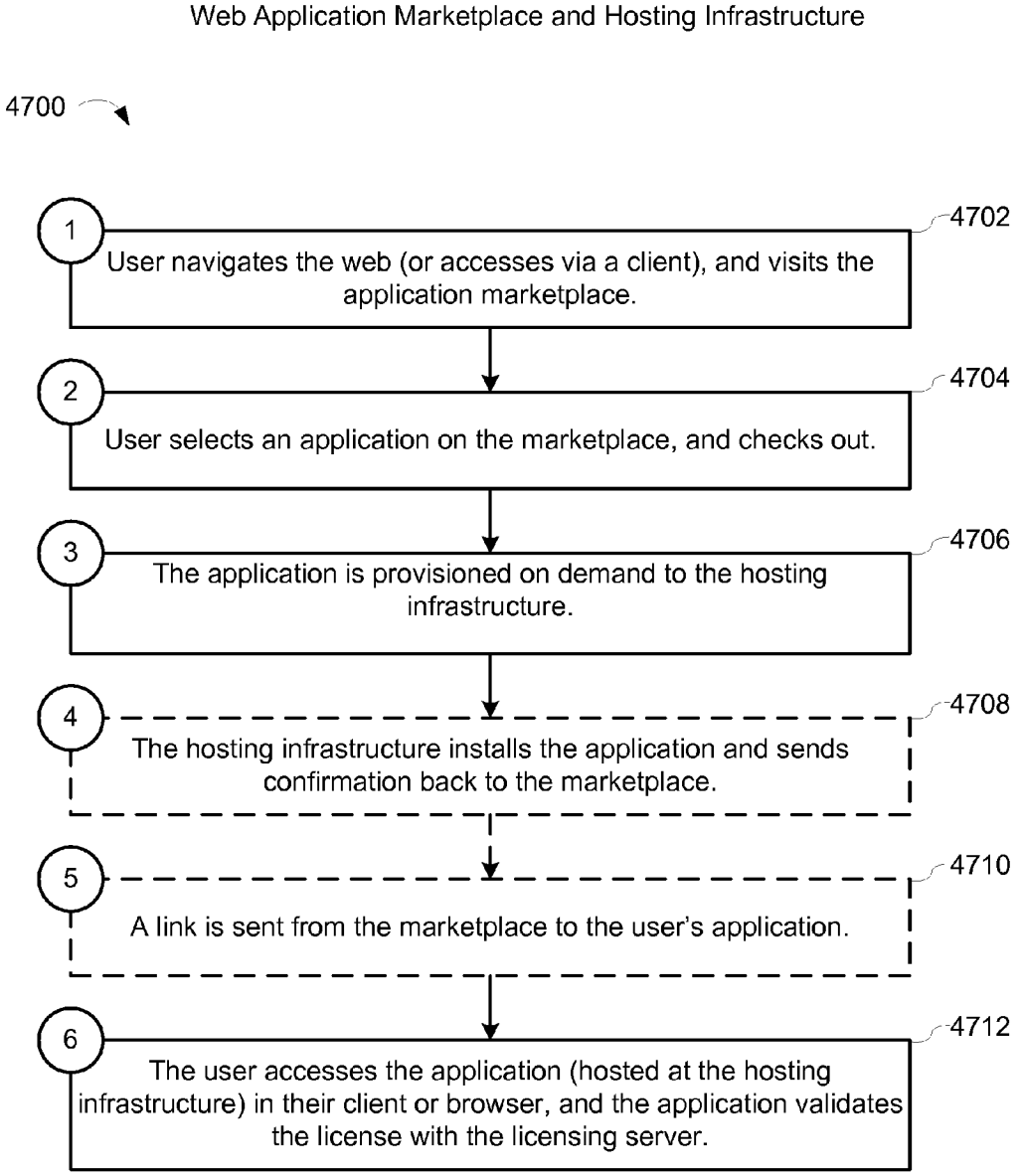

Web Application Marketplace and Hosting Infrastructure
4700
1
User navigates the web (or accesses via a client), and visits the application marketplace.
4702
2
User selects an application on the marketplace, and checks out.
4704
3
The application is provisioned on demand to the hosting infrastructure.
4706
4
The hosting infrastructure installs the application and sends confirmation back to the marketplace.
4708
5
A link is sent from the marketplace to the user's application.
4710
6
The user accesses the application (hosted at the hosting infrastructure) in their client or browser, and the application validates the license with the licensing server.
4712

Figure 47

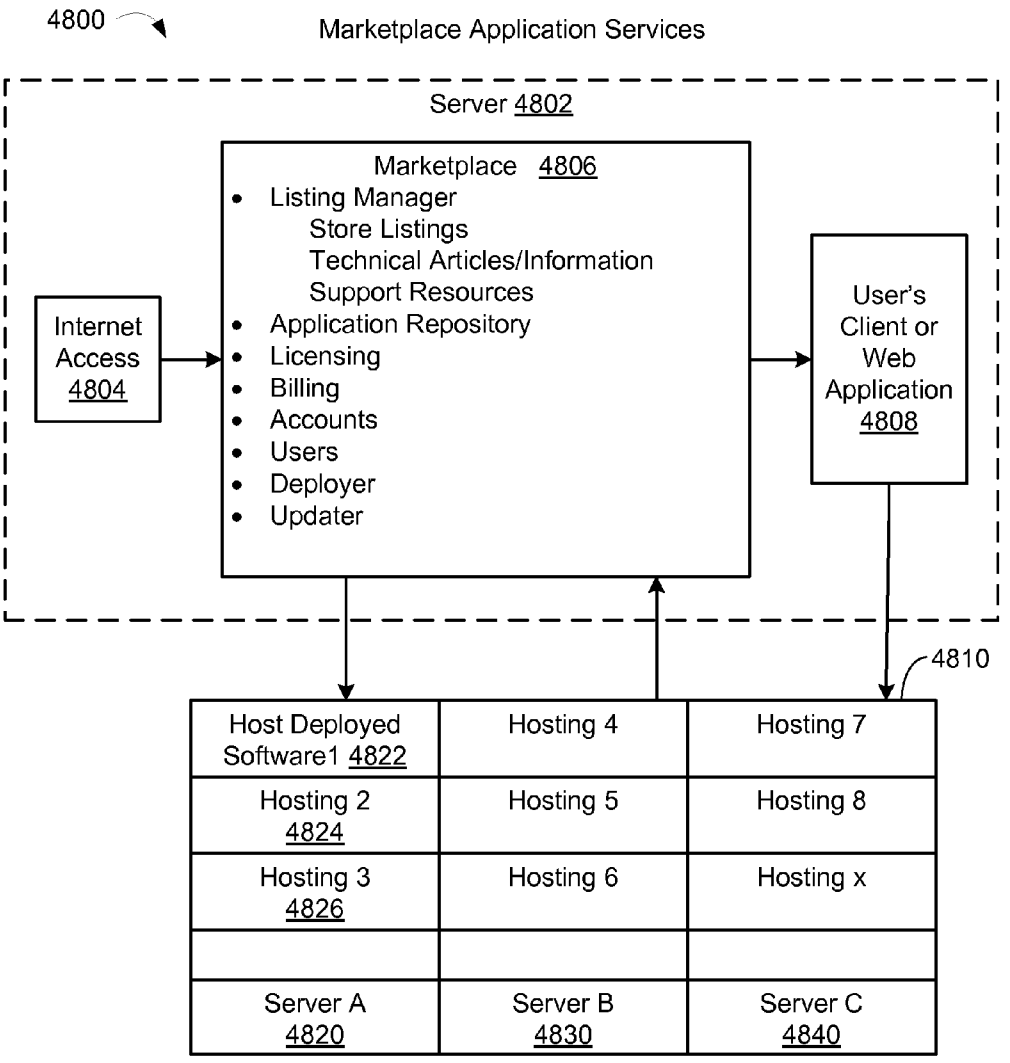

4800
Marketplace Application Services
Server 4802
Marketplace 4806
Listing Manager
Store Listings
Technical Articles/Information
Support Resources
Application Repository
Licensing
Billing
Accounts
Users
Deployer
Updater
Internet Access 4804
User's Client or Web Application 4808
4810
Host Deployed Software1 4822
Hosting 4
Hosting 7
Hosting 2 4824
Hosting 5
Hosting 8
Hosting 3 4826
Hosting 6
Hosting x
Server A 4820
Server B 4830
Server C 4840

Figure 48

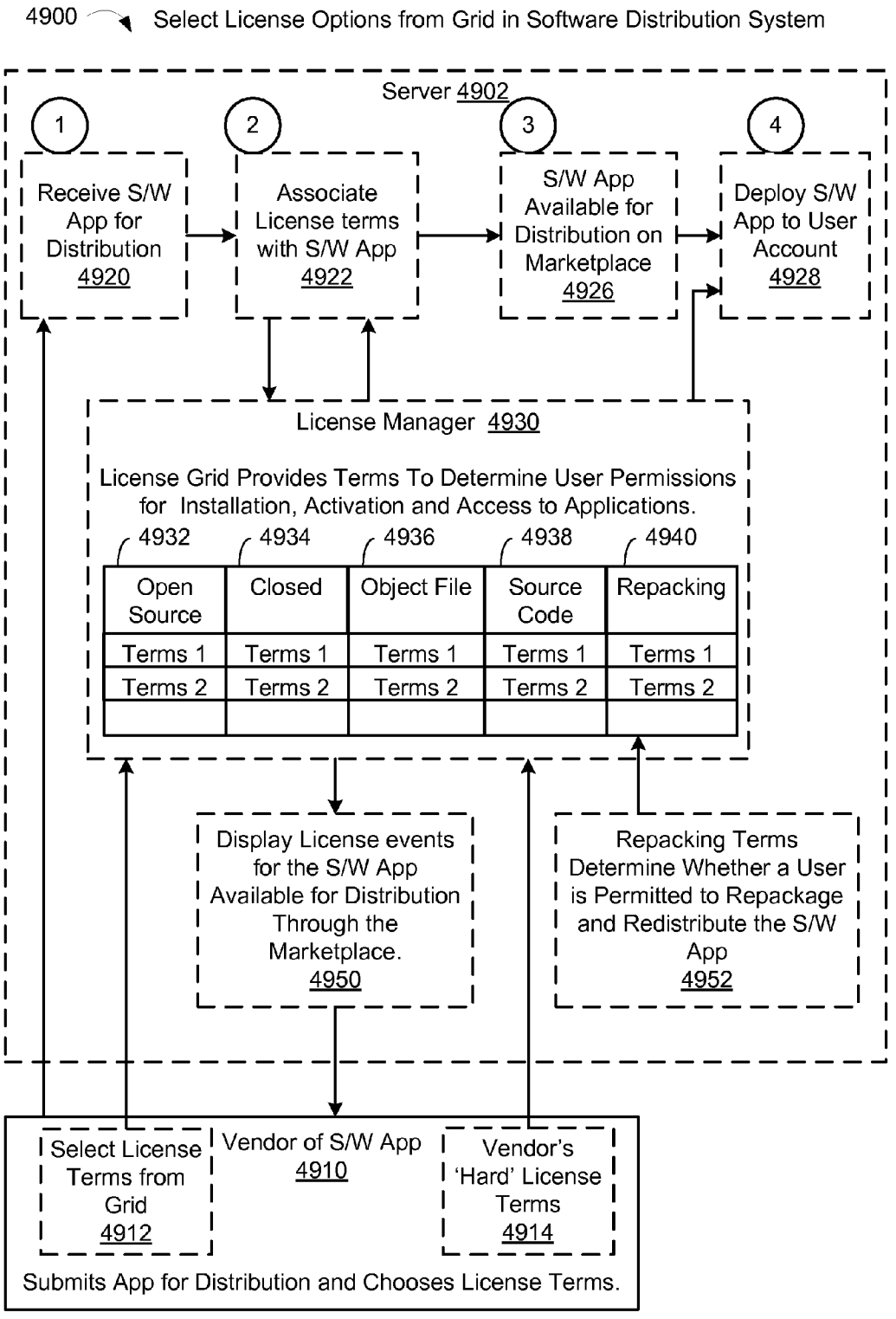

4900
Select License Options from Grid in Software Distribution System
Server 4902
1
2
3
4
Receive S/W App for Distribution 4920
Associate License terms with S/W App 4922
S/W App Available for Distribution on Marketplace 4926
Deploy S/W App to User Account 4928
License Manager 4930
License Grid Provides Terms To Determine User Permissions for Installation, Activation and Access to Applications.
4932
4934
4936
4938
4940
Open Source
Closed
Object File
Source Code
Repacking
Terms 1
Terms 2
Display License events for the S/W App Available for Distribution Through the Marketplace. 4950
Repacking Terms Determine Whether a User is Permitted to Repackage and Redistribute the S/W App 4952
Select License Terms from Grid 4912
Vendor of S/W App 4910
Vendor's 'Hard' License Terms 4914
Submits App for Distribution and Chooses License Terms.

Figure 49

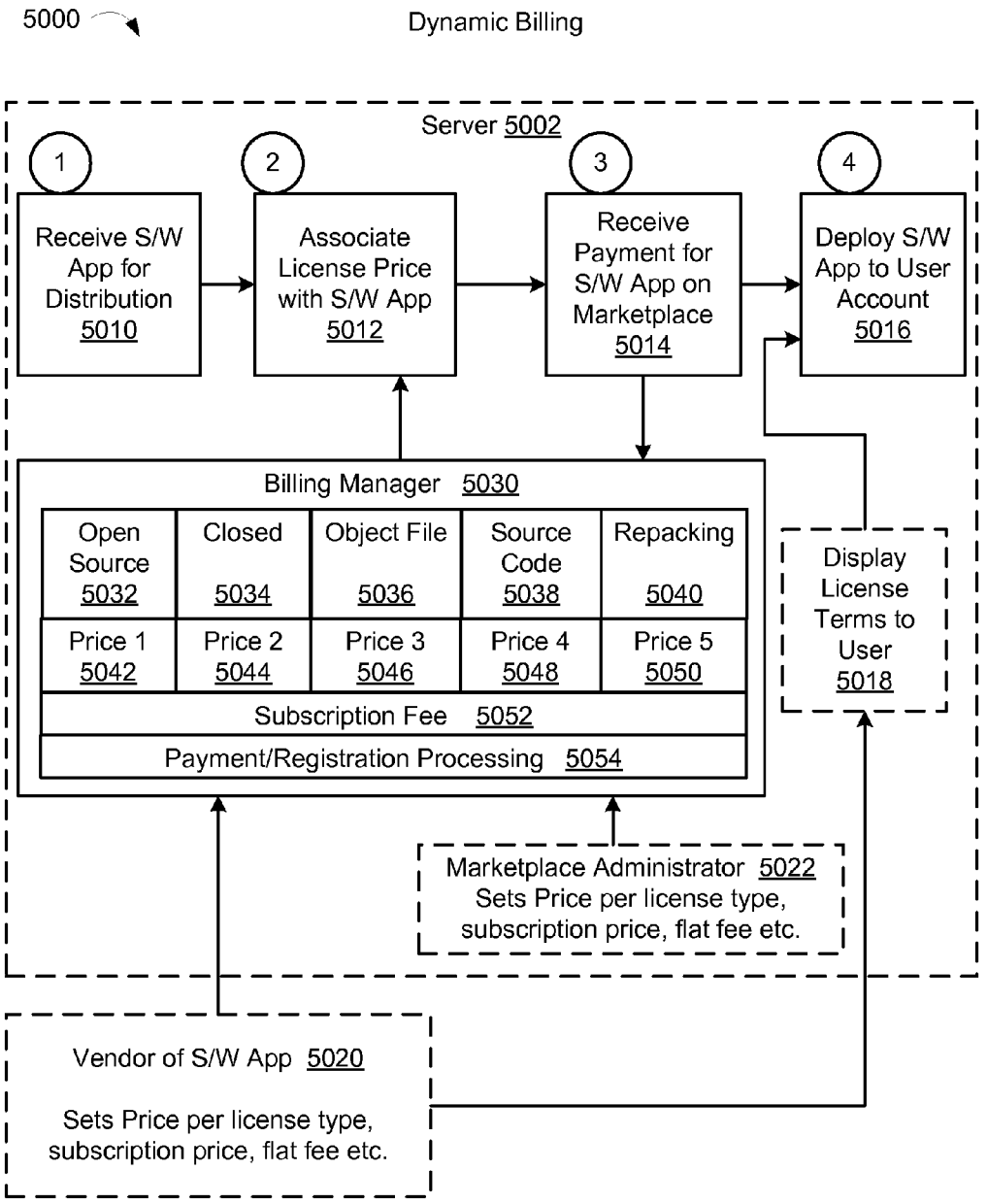

5000
Dynamic Billing
Server 5002
1
2
3
4
Receive S/W App for Distribution 5010
Associate License Price with S/W App 5012
Receive Payment for S/W App on Marketplace 5014
Deploy S/W App to User Account 5016
Billing Manager 5030
Open Source 5032
Closed 5034
Object File 5036
Source Code 5038
Repacking 5040
Price 1 5042
Price 2 5044
Price 3 5046
Price 4 5048
Price 5 5050
Subscription Fee 5052
Payment/Registration Processing 5054
Display License Terms to User 5018
Marketplace Administrator 5022
Sets Price per license type, subscription price, flat fee etc.
Vendor of S/W App 5020
Sets Price per license type, subscription price, flat fee etc.

Figure 50

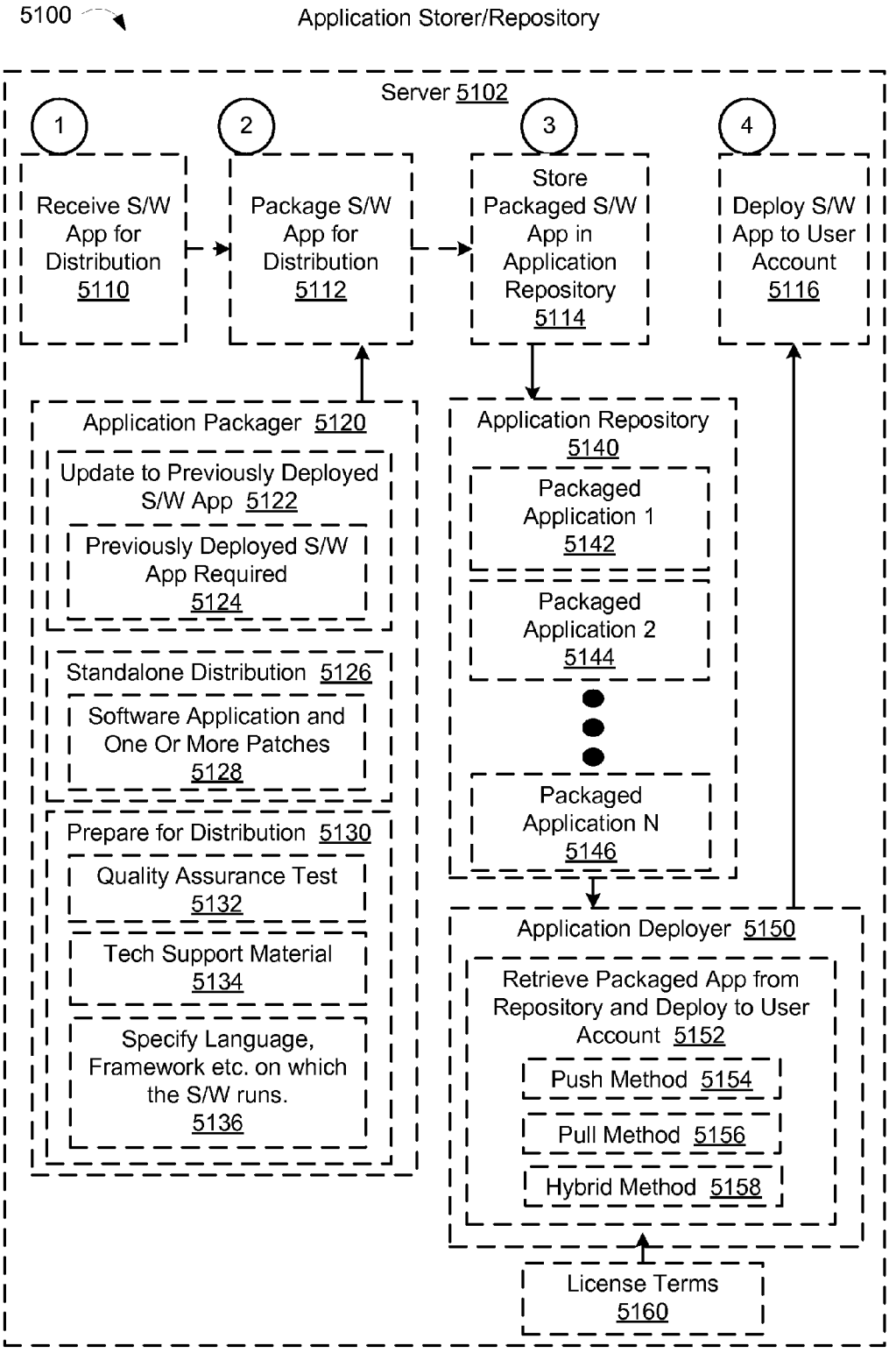

5100
Application Storer/Repository
Server 5102
1
2
3
4
Receive S/W App for Distribution 5110
Package S/W App for Distribution 5112
Store Packaged S/W App in Application Repository 5114
Deploy S/W App to User Account 5116
Application Packager 5120
Update to Previously Deployed S/W App 5122
Previously Deployed S/W App Required 5124
Standalone Distribution 5126
Software Application and One Or More Patches 5128
Prepare for Distribution 5130
Quality Assurance Test 5132
Tech Support Material 5134
Specify Language, Framework etc. on which the S/W runs. 5136
Application Repository 5140
Packaged Application 1 5142
Packaged Application 2 5144
Packaged Application N 5146
Application Deployer 5150
Retrieve Packaged App from Repository and Deploy to User Account 5152
Push Method 5154
Pull Method 5156
Hybrid Method 5158
License Terms 5160

Figure 51

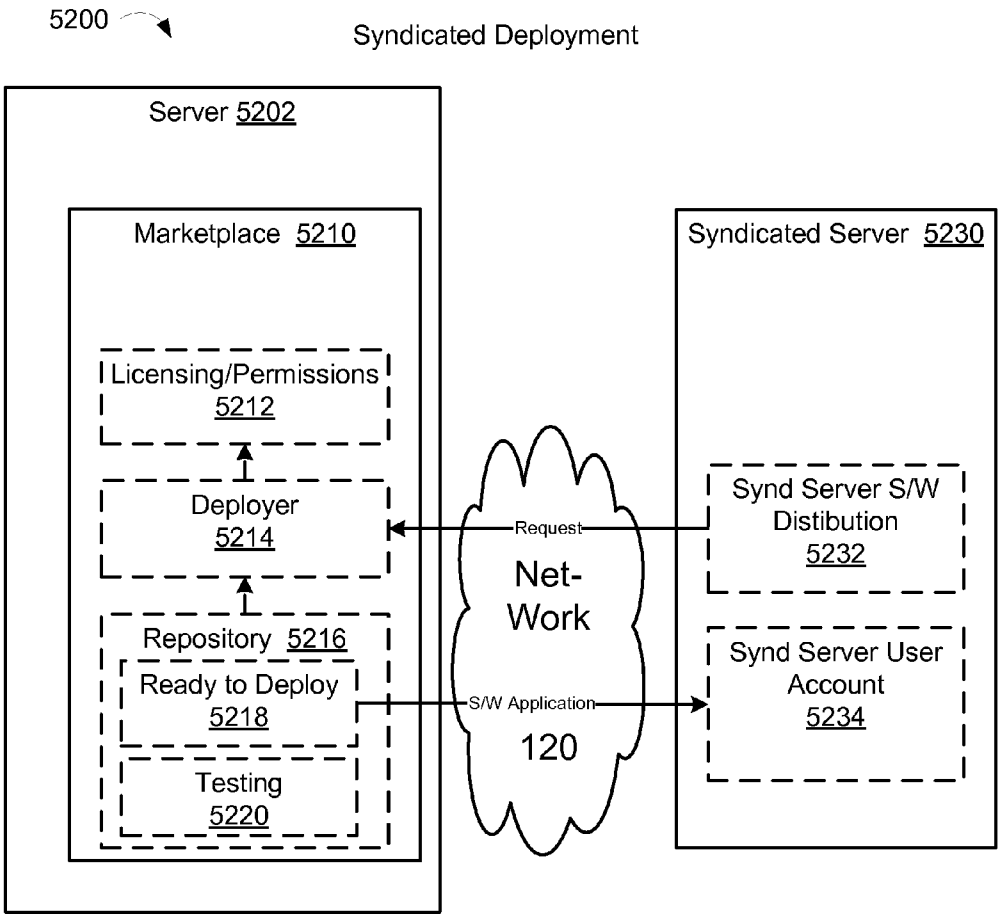

5200
Syndicated Deployment
Server 5202
Marketplace 5210
Licensing/Permissions 5212
Deployer 5214
Repository 5216
Ready to Deploy 5218
Testing 5220
Request
S/W Application
Net-Work
120
Syndicated Server 5230
Synd Server S/W Distibution 5232
Synd Server User Account 5234

Figure 52

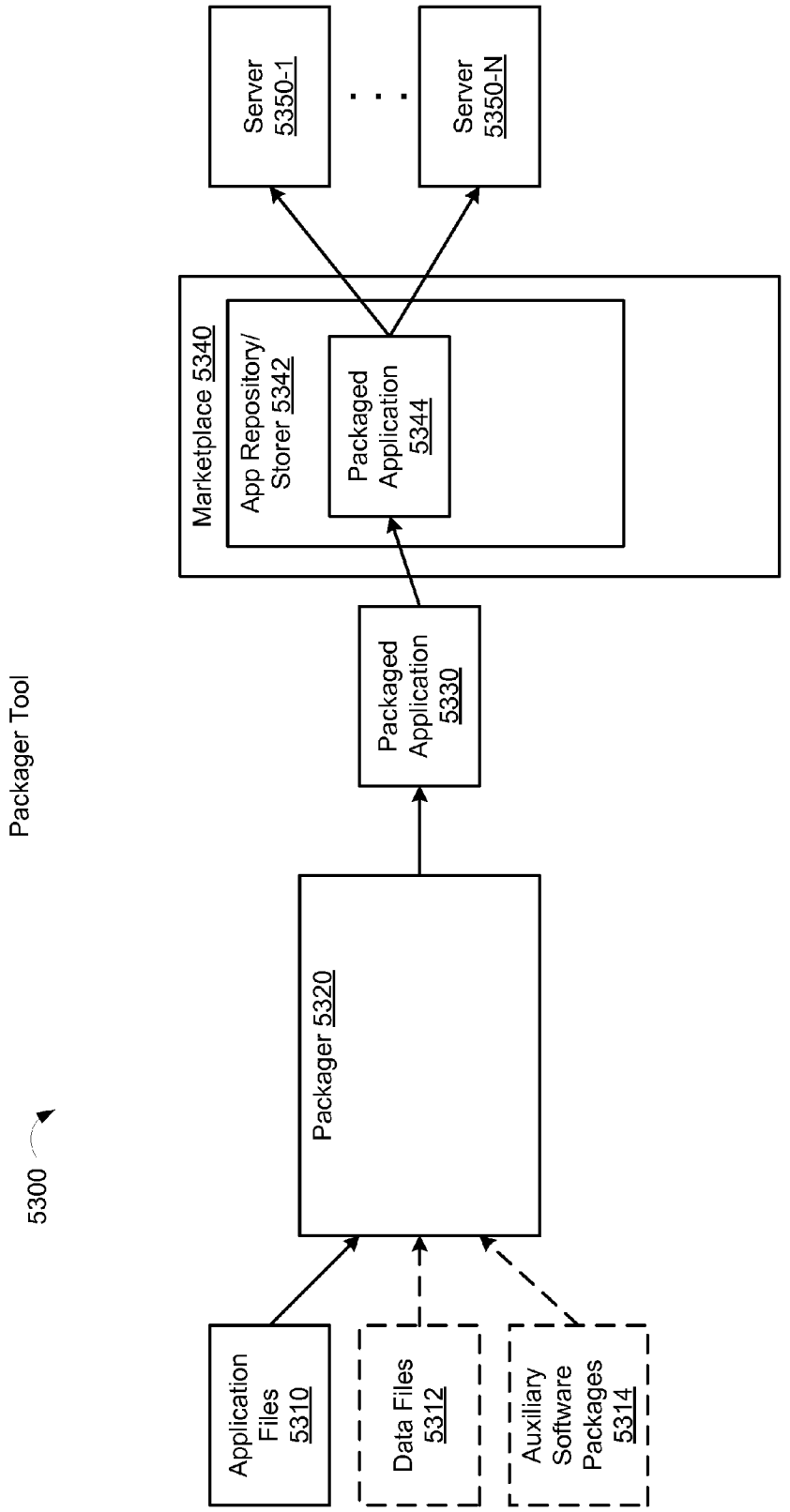

Packager Tool
5300
Application Files 5310
Data Files 5312
Auxiliary Software Packages 5314
Packager 5320
Packaged Application 5330
Marketplace 5340
App Repository/ Storer 5342
Packaged Application 5344
Server 5350-1
Server 5350-N

Figure 53

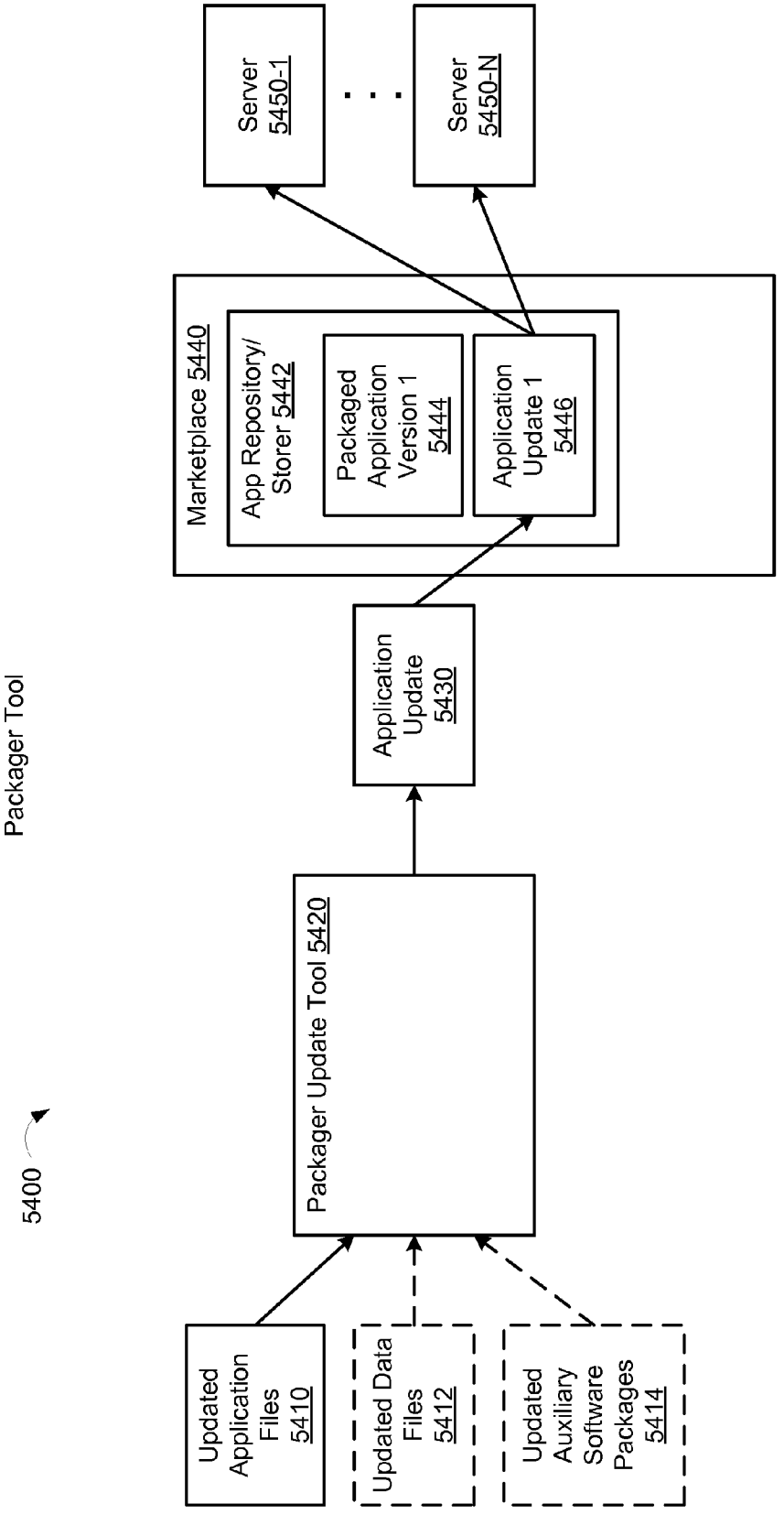

Packager Tool
5400
Updated Application Files 5410
Updated Data Files 5412
Updated Auxiliary Software Packages 5414
Packager Update Tool 5420
Application Update 5430
Marketplace 5440
App Repository/ Storer 5442
Packaged Application Version 1 5444
Application Update 1 5446
Server 5450-1
Server 5450-N

Figure 54

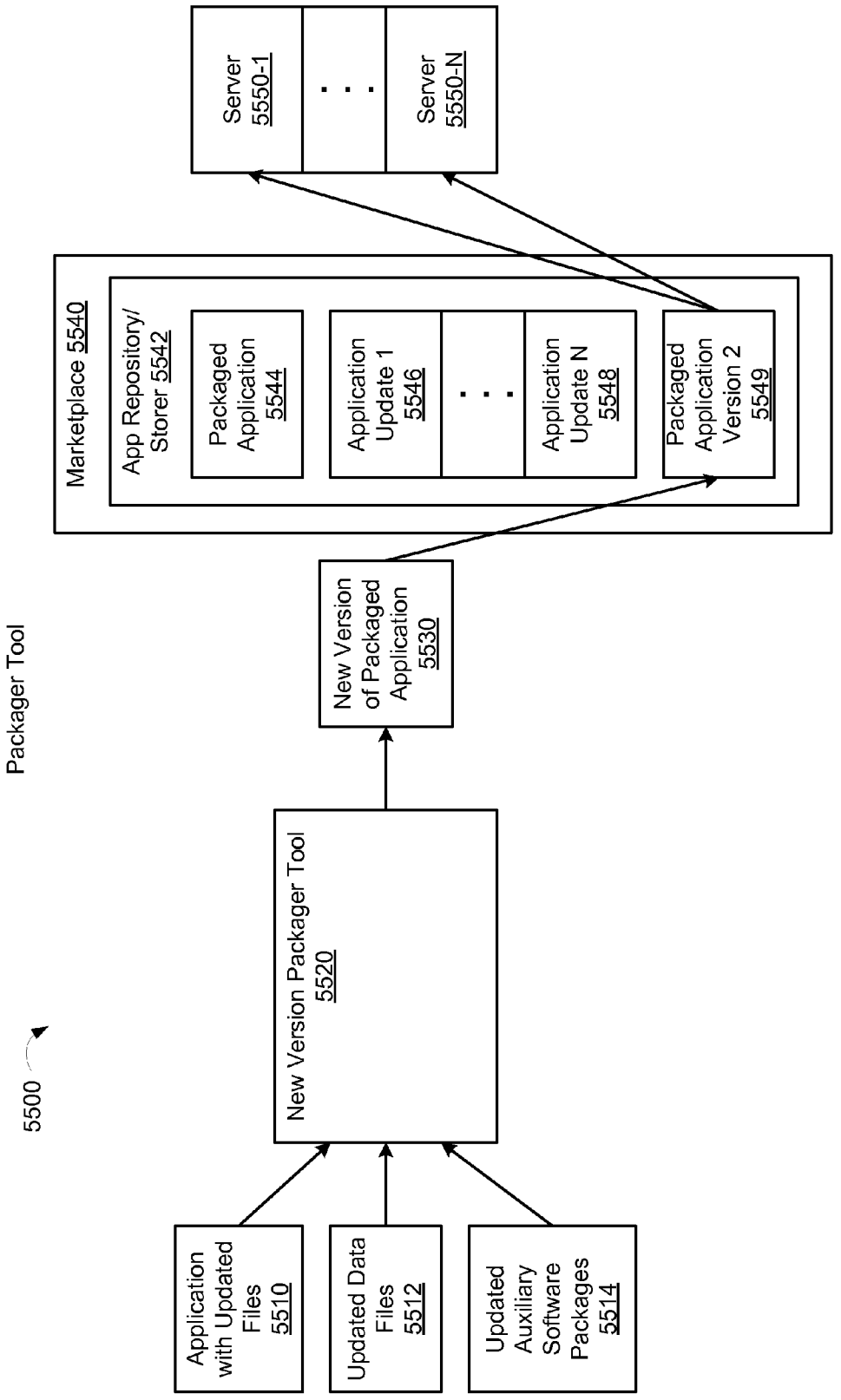

5500
Packager Tool
Application with Updated Files 5510
Updated Data Files 5512
Updated Auxiliary Software Packages 5514
New Version Packager Tool 5520
New Version of Packaged Application 5530
Marketplace 5540
App Repository/ Storer 5542
Packaged Application 5544
Application Update 1 5546
Application Update N 5548
Packaged Application Version 2 5549
Server 5550-1
Server 5550-N

Figure 55

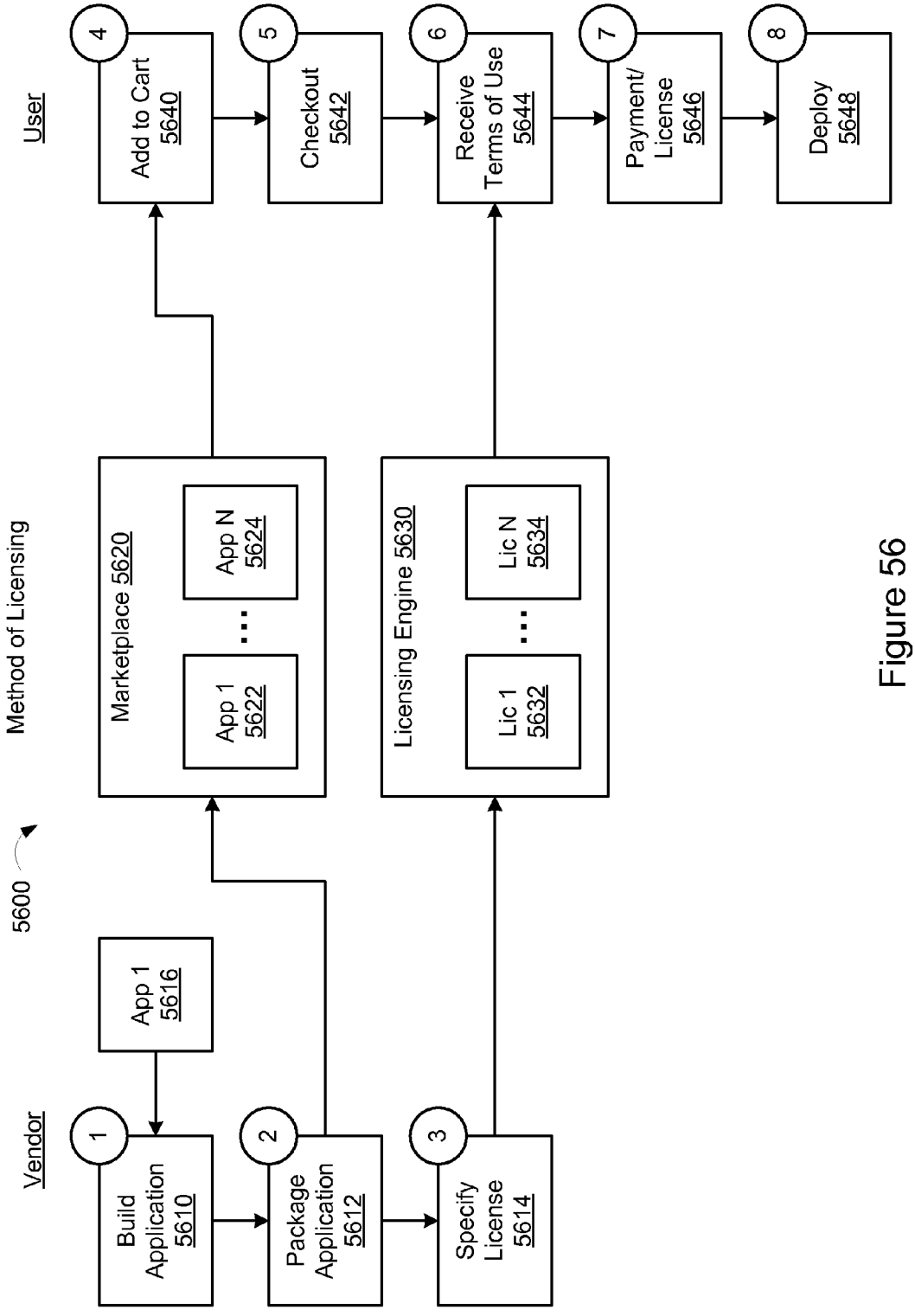

Method of Licensing
5600
Vendor
User
1
Build Application 5610
2
Package Application 5612
3
Specify License 5614
App 1 5616
Marketplace 5620
App 1 5622
App N 5624
Licensing Engine 5630
Lic 1 5632
Lic N 5634
4
Add to Cart 5640
5
Checkout 5642
6
Receive Terms of Use 5644
7
Payment/ License 5646
8
Deploy 5648

Figure 56

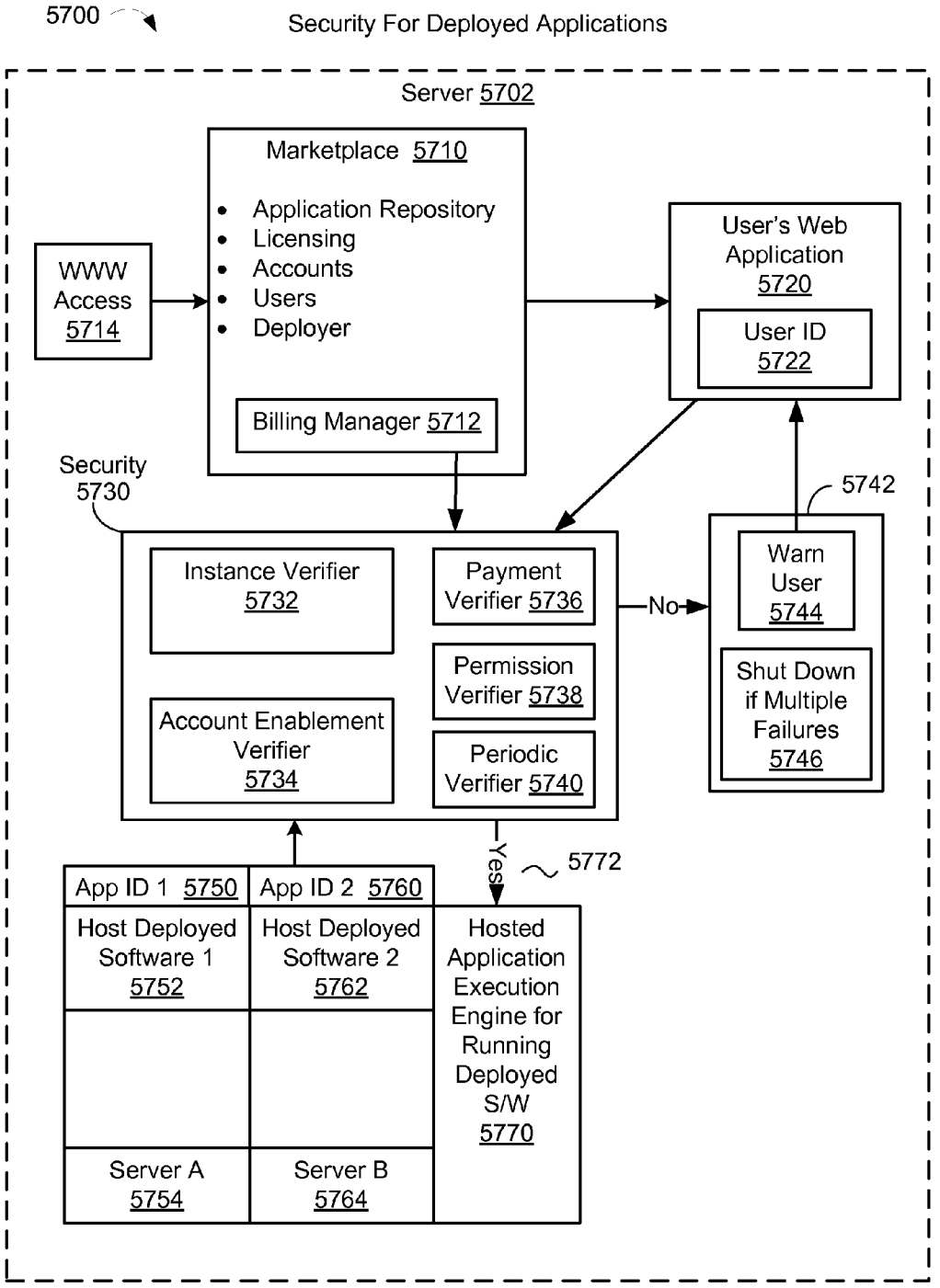

5700
Security For Deployed Applications
Server 5702
Marketplace 5710
Application Repository
Licensing
Accounts
Users
Deployer
Billing Manager 5712
WWW Access 5714
User's Web Application 5720
User ID 5722
Security 5730
5742
Instance Verifier 5732
Payment Verifier 5736
No
Warn User 5744
Permission Verifier 5738
Shut Down if Multiple Failures 5746
Account Enablement Verifier 5734
Periodic Verifier 5740
Yes
5772
App ID 1 5750
App ID 2 5760
Host Deployed Software 1 5752
Host Deployed Software 2 5762
Hosted Application Execution Engine for Running Deployed S/W 5770
Server A 5754
Server B 5764

Figure 57

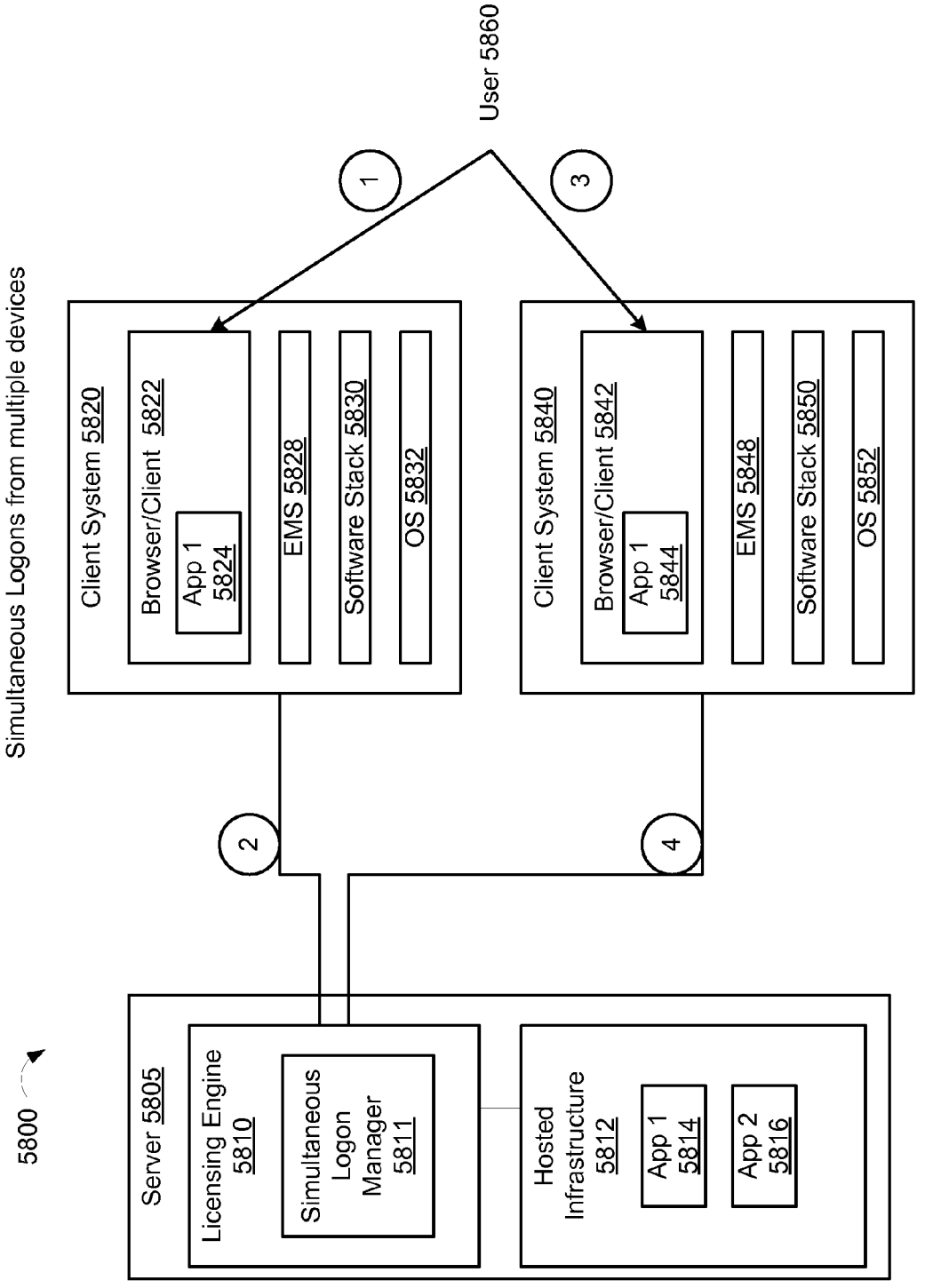

Simultaneous Logons from multiple devices
5800
User 5860
1
2
3
4
Client System 5820
Browser/Client 5822
App 1 5824
EMS 5828
Software Stack 5830
OS 5832
Client System 5840
Browser/Client 5842
App 1 5844
EMS 5848
Software Stack 5850
OS 5852
Server 5805
Licensing Engine 5810
Simultaneous Logon Manager 5811
Hosted Infrastructure 5812
App 1 5814
App 2 5816

Figure 58

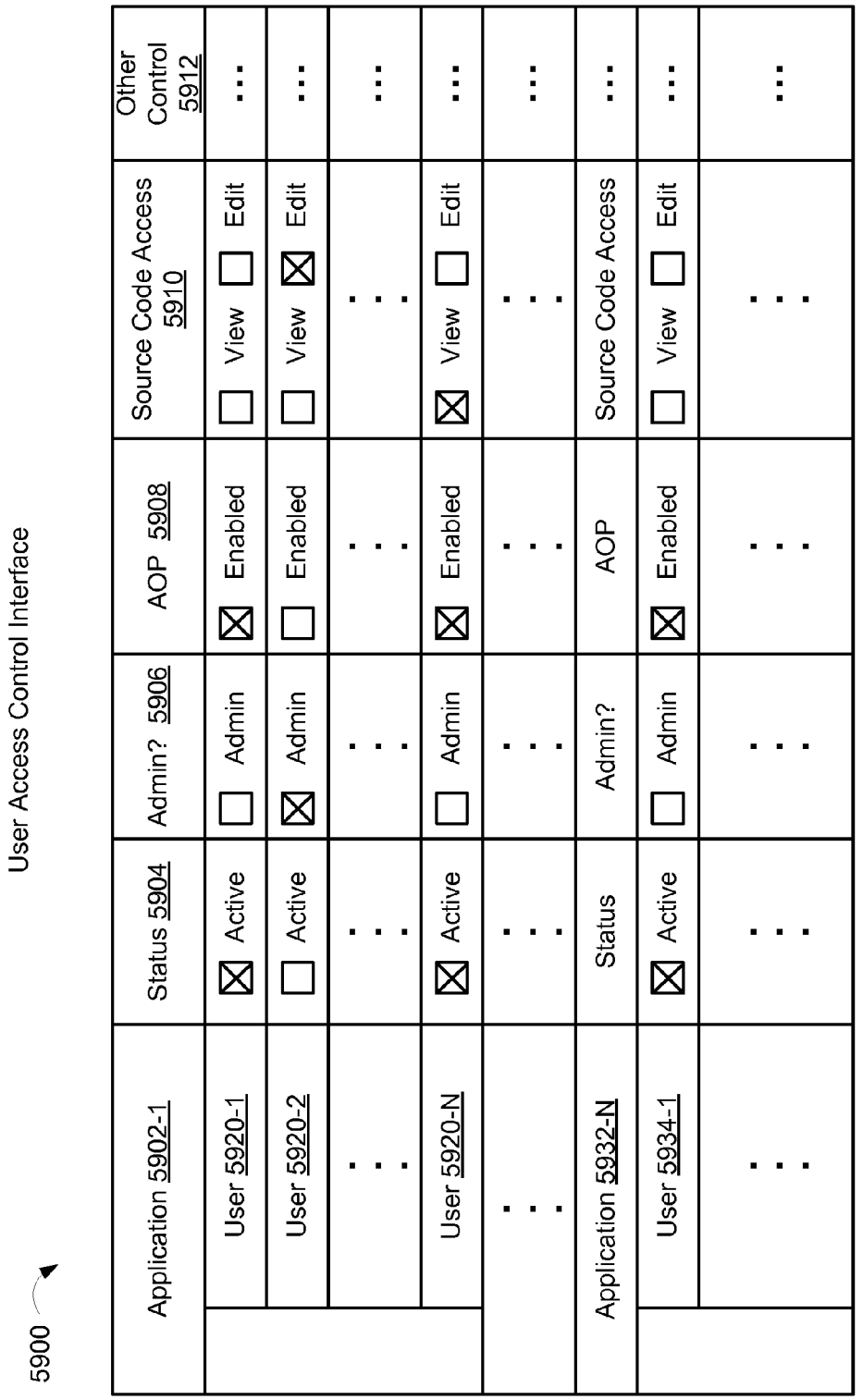

5900
User Access Control Interface
Application 5902-1
Status 5904
Admin? 5906
AOP 5908
Source Code Access 5910
Other Control 5912
User 5920-1
Active
Admin
Enabled
View
Edit
User 5920-2
Active
Admin
Enabled
View
Edit
User 5920-N
Active
Admin
Enabled
View
Edit
Application 5932-N
Status
Admin?
AOP
Source Code Access
User 5934-1
Active
Admin
Enabled
View
Edit

Figure 59

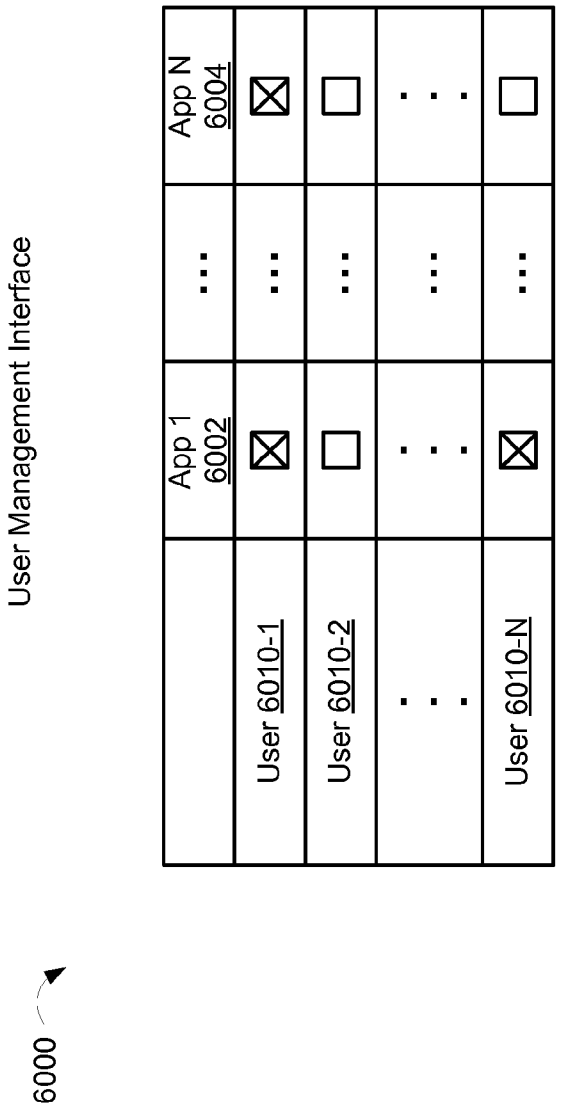

6000
User Management Interface
App 1 6002
App N 6004
User 6010-1
User 6010-2
User 6010-N

Figure 60

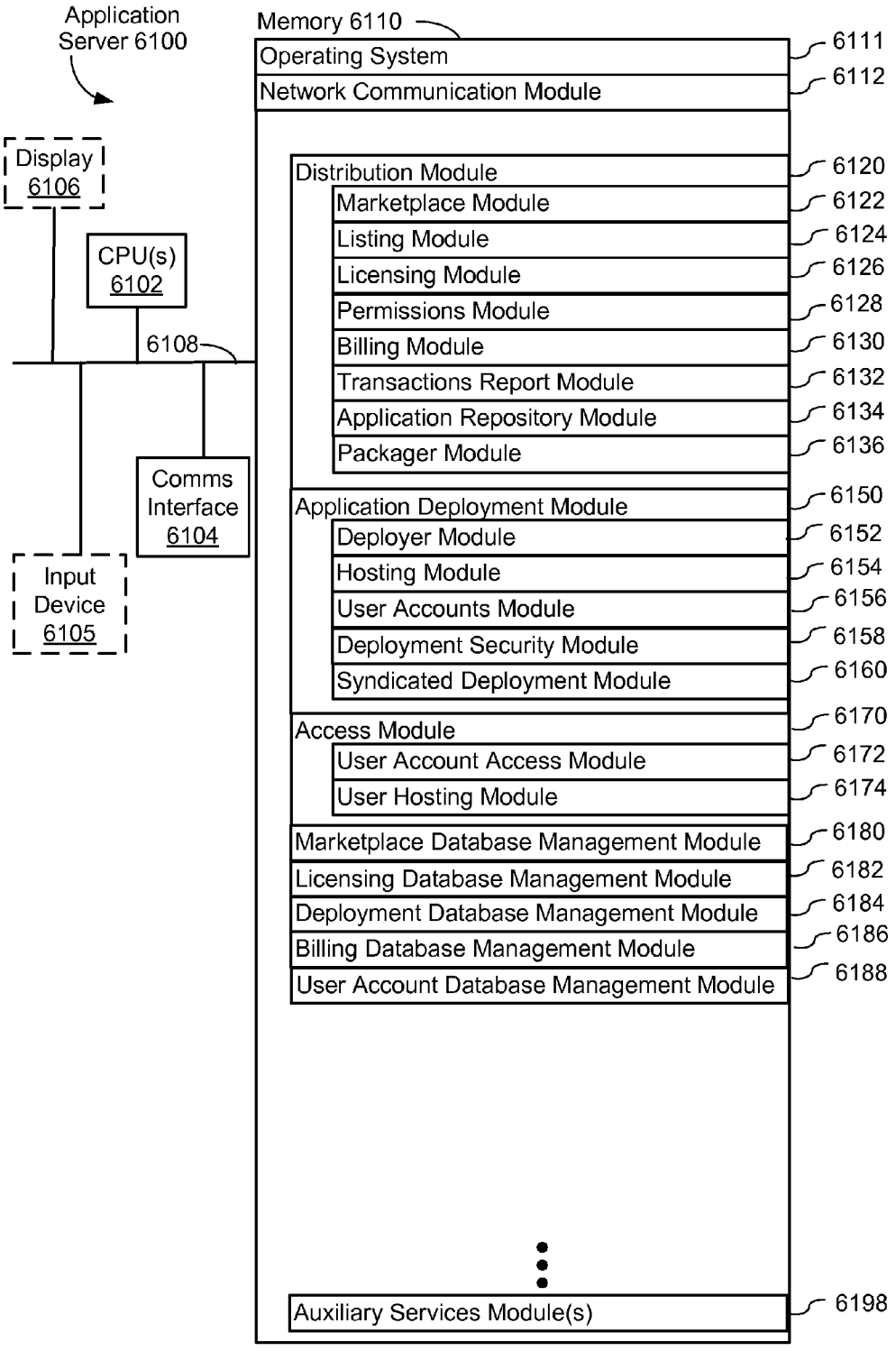

Application Server 6100
Memory 6110
Operating System
6111
Network Communication Module
6112
Display 6106
CPU(s) 6102
6108
Comms Interface 6104
Input Device 6105
Distribution Module
6120
Marketplace Module
6122
Listing Module
6124
Licensing Module
6126
Permissions Module
6128
Billing Module
6130
Transactions Report Module
6132
Application Repository Module
6134
Packager Module
6136
Application Deployment Module
6150
Deployer Module
6152
Hosting Module
6154
User Accounts Module
6156
Deployment Security Module
6158
Syndicated Deployment Module
6160
Access Module
6170
User Account Access Module
6172
User Hosting Module
6174
Marketplace Database Management Module
6180
Licensing Database Management Module
6182
Deployment Database Management Module
6184
Billing Database Management Module
6186
User Account Database Management Module
6188
Auxiliary Services Module(s)
6198

Figure 61

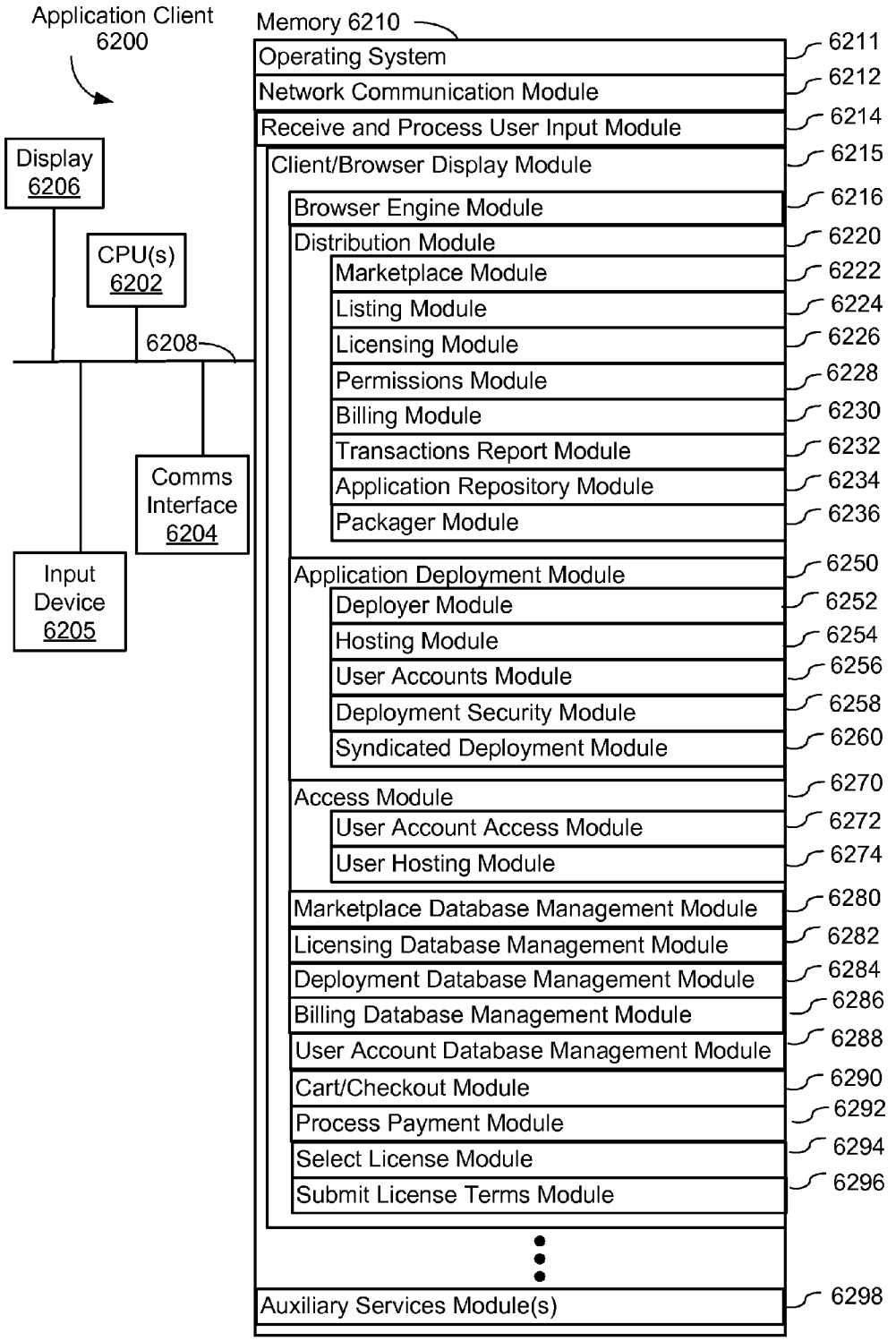

Application Client 6200
Memory 6210
Operating System
Network Communication Module
Receive and Process User Input Module
Client/Browser Display Module
Browser Engine Module
Distribution Module
Marketplace Module
Listing Module
Licensing Module
Permissions Module
Billing Module
Transactions Report Module
Application Repository Module
Packager Module
Application Deployment Module
Deployer Module
Hosting Module
User Accounts Module
Deployment Security Module
Syndicated Deployment Module
Access Module
User Account Access Module
User Hosting Module
Marketplace Database Management Module
Licensing Database Management Module
Deployment Database Management Module
Billing Database Management Module
User Account Database Management Module
Cart/Checkout Module
Process Payment Module
Select License Module
Submit License Terms Module
Auxiliary Services Module(s)
6211
6212
6214
6215
6216
6220
6222
6224
6226
6228
6230
6232
6234
6236
6250
6252
6254
6256
6258
6260
6270
6272
6274
6280
6282
6284
6286
6288
6290
6292
6294
6296
6298
Display 6206
CPU(s) 6202
6208
Comms Interface 6204
Input Device 6205

Figure 62

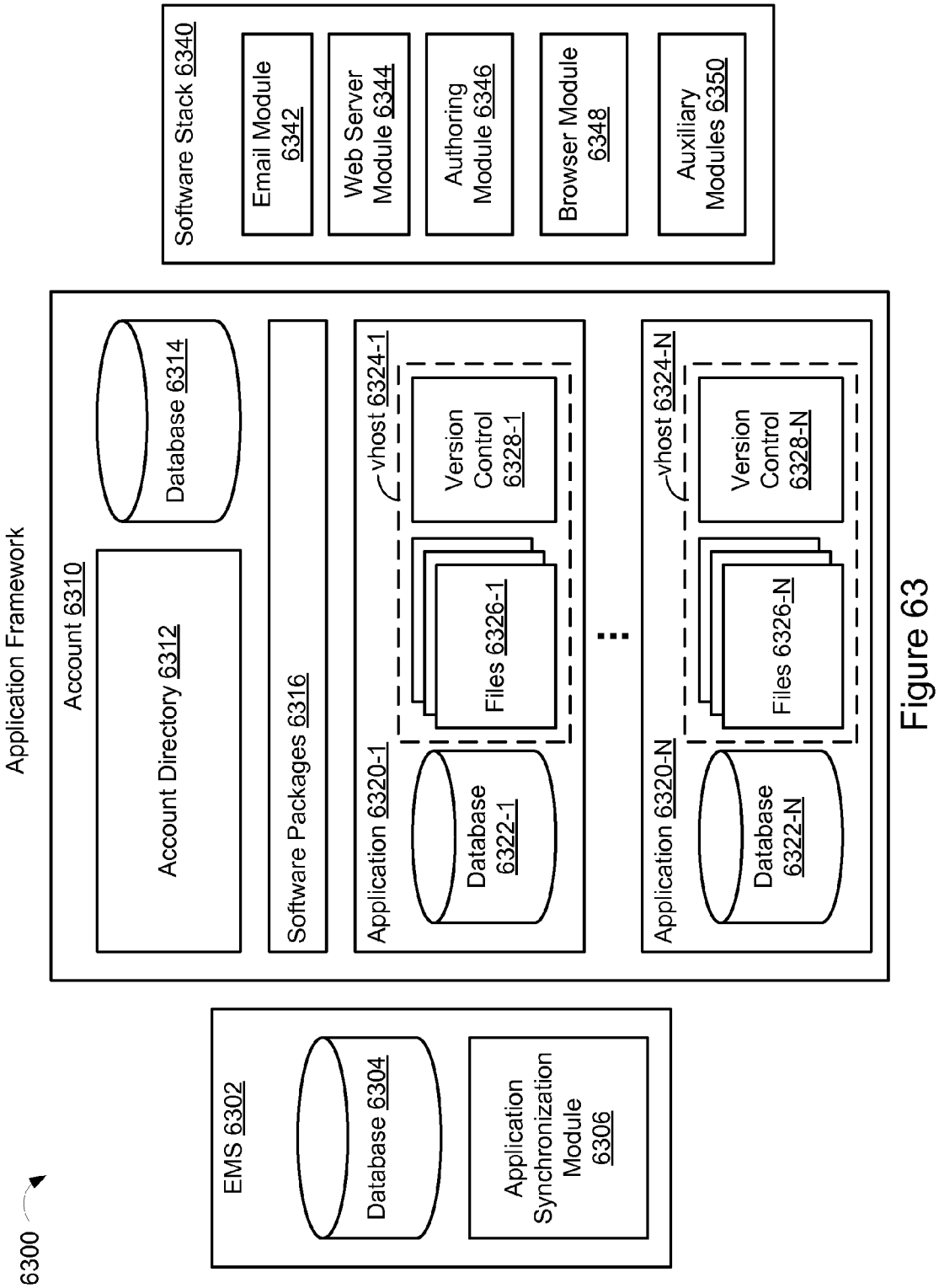

6300
EMS 6302
Database 6304
Application Synchronization Module 6306
Application Framework
Account 6310
Account Directory 6312
Database 6314
Software Packages 6316
Application 6320-1
vhost 6324-1
Version Control 6328-1
Files 6326-1
Database 6322-1
Application 6320-N
vhost 6324-N
Version Control 6328-N
Files 6326-N
Database 6322-N
Software Stack 6340
Email Module 6342
Web Server Module 6344
Authoring Module 6346
Browser Module 6348
Auxiliary Modules 6350

Figure 63

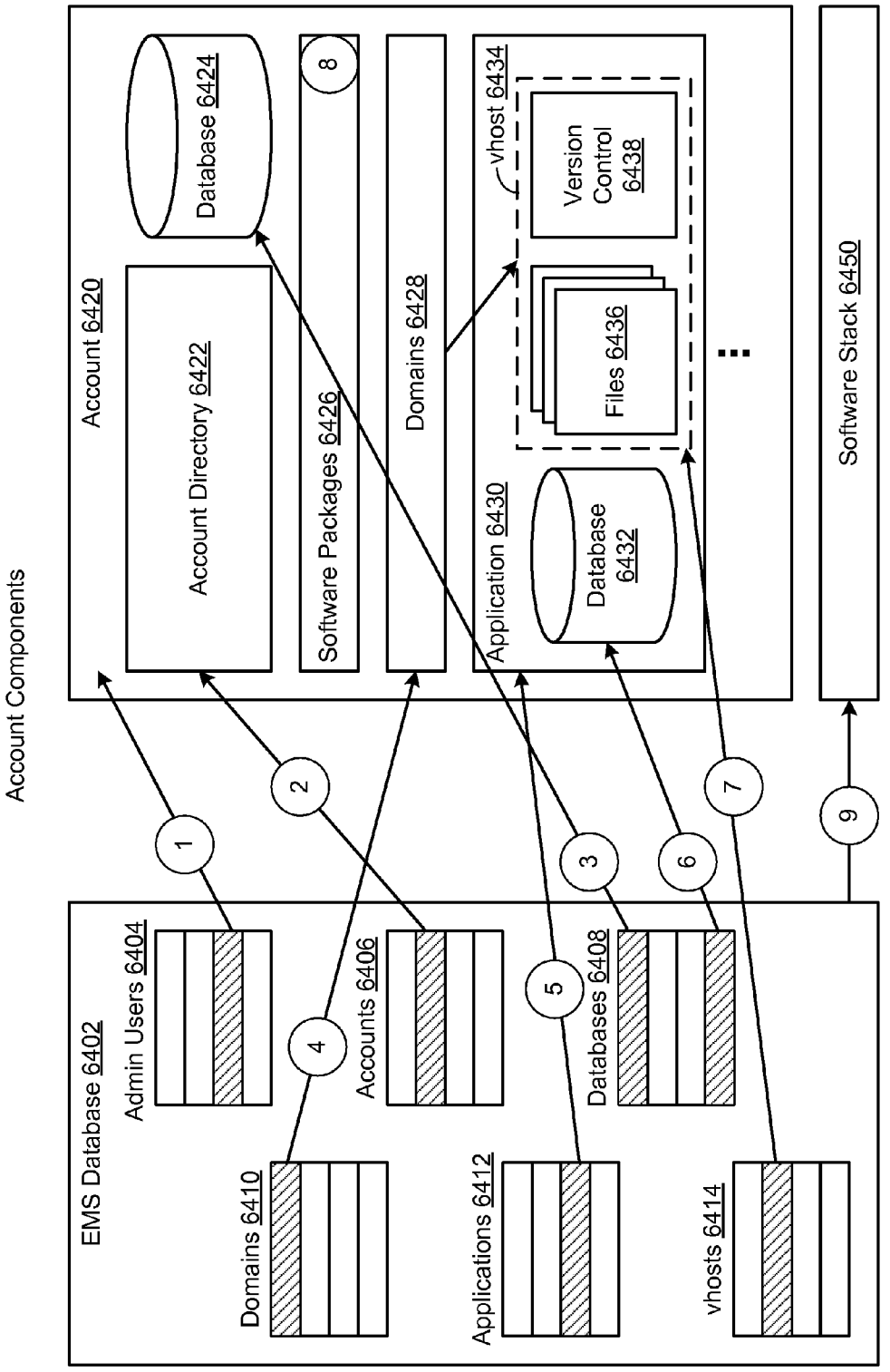

Account Components
EMS Database 6402
Admin Users 6404
Accounts 6406
Databases 6408
Domains 6410
Applications 6412
vhosts 6414
Account 6420
Account Directory 6422
Database 6424
Software Packages 6426
Domains 6428
Application 6430
Database 6432
vhost 6434
Version Control 6438
Files 6436
Software Stack 6450
1
2
3
4
5
6
7
8
9

Figure 64A

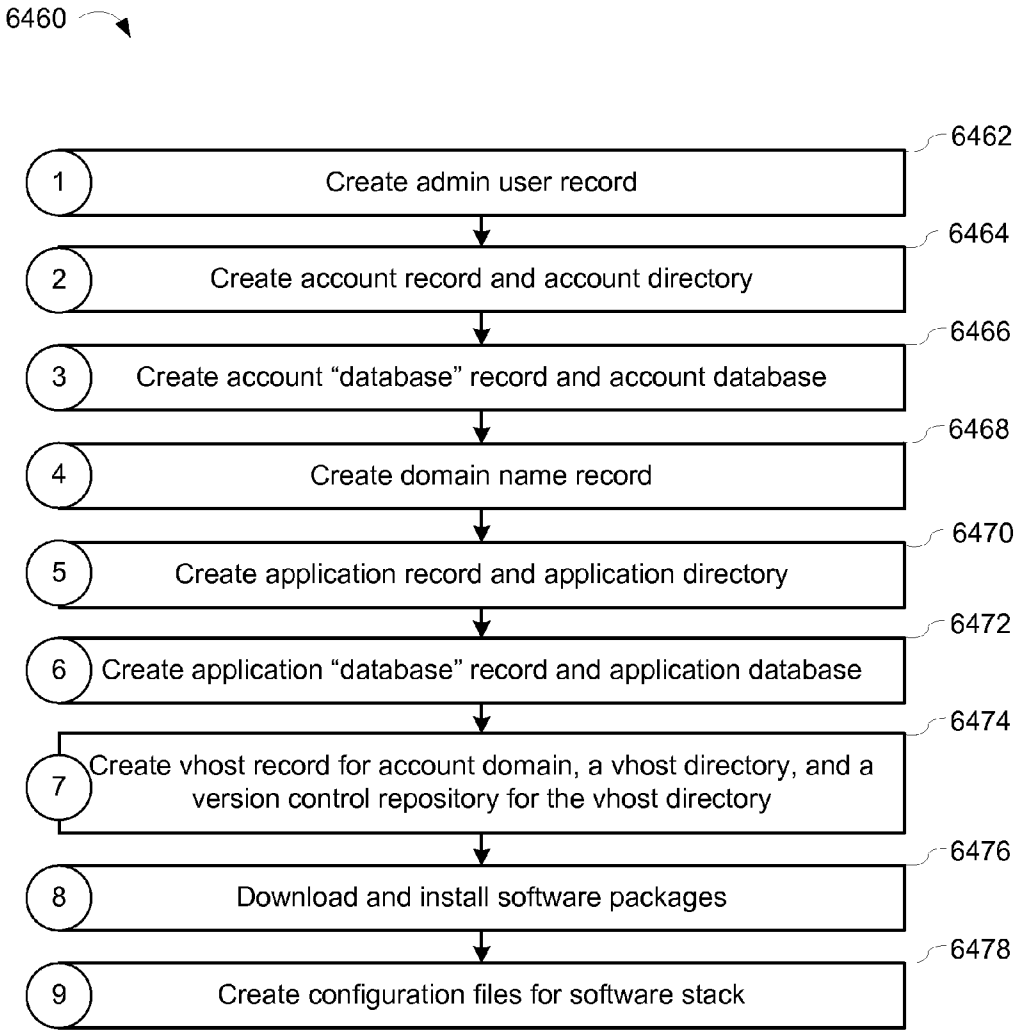

6460
6462
1
Create admin user record
6464
2
Create account record and account directory
6466
3
Create account "database" record and account database
6468
4
Create domain name record
6470
5
Create application record and application directory
6472
6
Create application "database" record and application database
6474
7
Create vhost record for account domain, a vhost directory, and a version control repository for the vhost directory
6476
8
Download and install software packages
6478
9
Create configuration files for software stack

Figure 64B

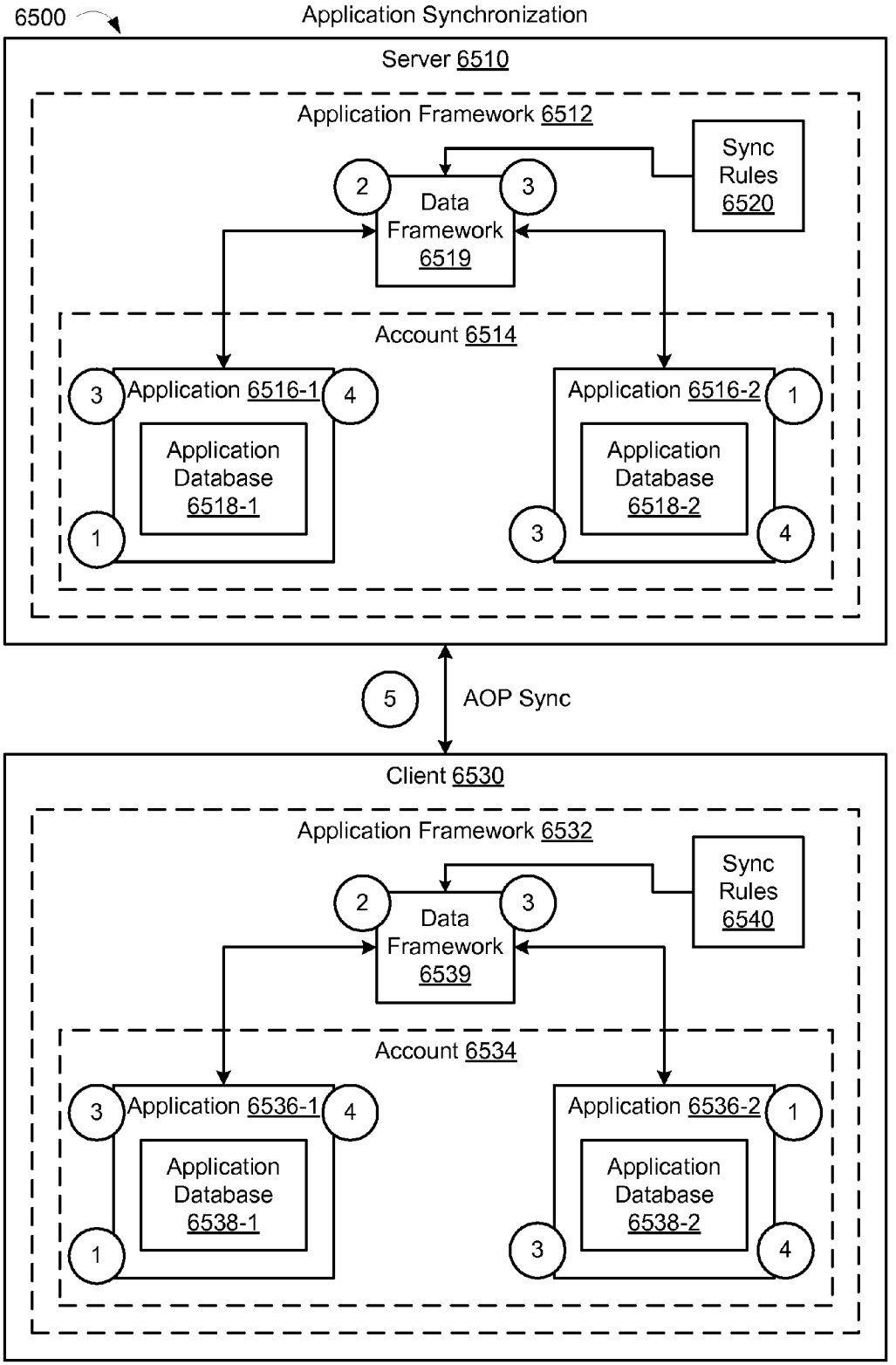

6500
Application Synchronization
Server 6510
Application Framework 6512
Sync Rules 6520
2
3
Data Framework 6519
Account 6514
3
Application 6516-1
4
Application Database 6518-1
1
Application 6516-2
1
Application Database 6518-2
3
4
5
AOP Sync
Client 6530
Application Framework 6532
Sync Rules 6540
2
3
Data Framework 6539
Account 6534
3
Application 6536-1
4
Application Database 6538-1
1
Application 6536-2
1
Application Database 6538-2
3
4

Figure 65

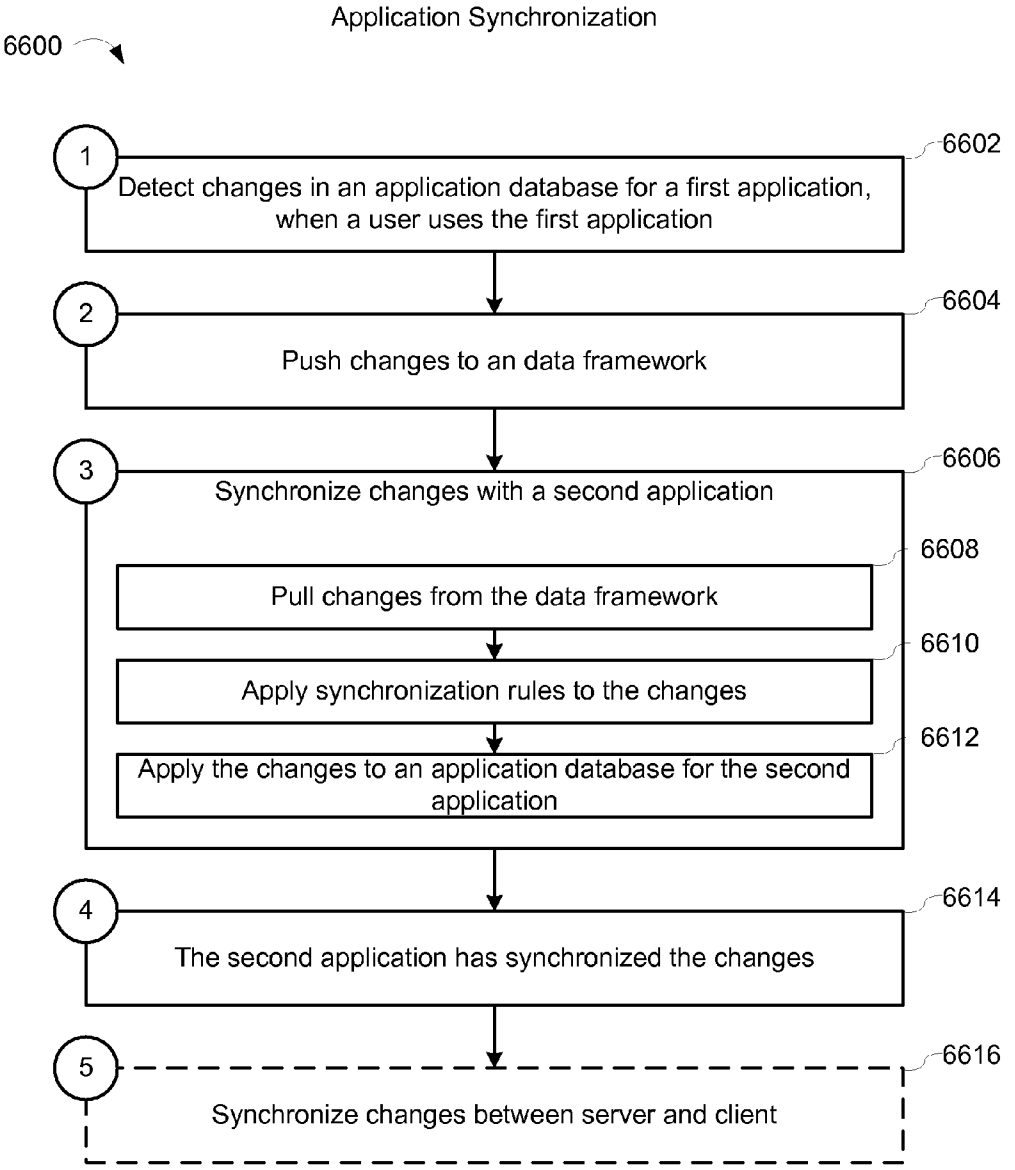

Application Synchronization
6600
1
Detect changes in an application database for a first application, when a user uses the first application
6602
2
Push changes to an data framework
6604
3
Synchronize changes with a second application
6606
Pull changes from the data framework
6608
Apply synchronization rules to the changes
6610
Apply the changes to an application database for the second application
6612
4
The second application has synchronized the changes
6614
5
Synchronize changes between server and client
6616

Figure 66

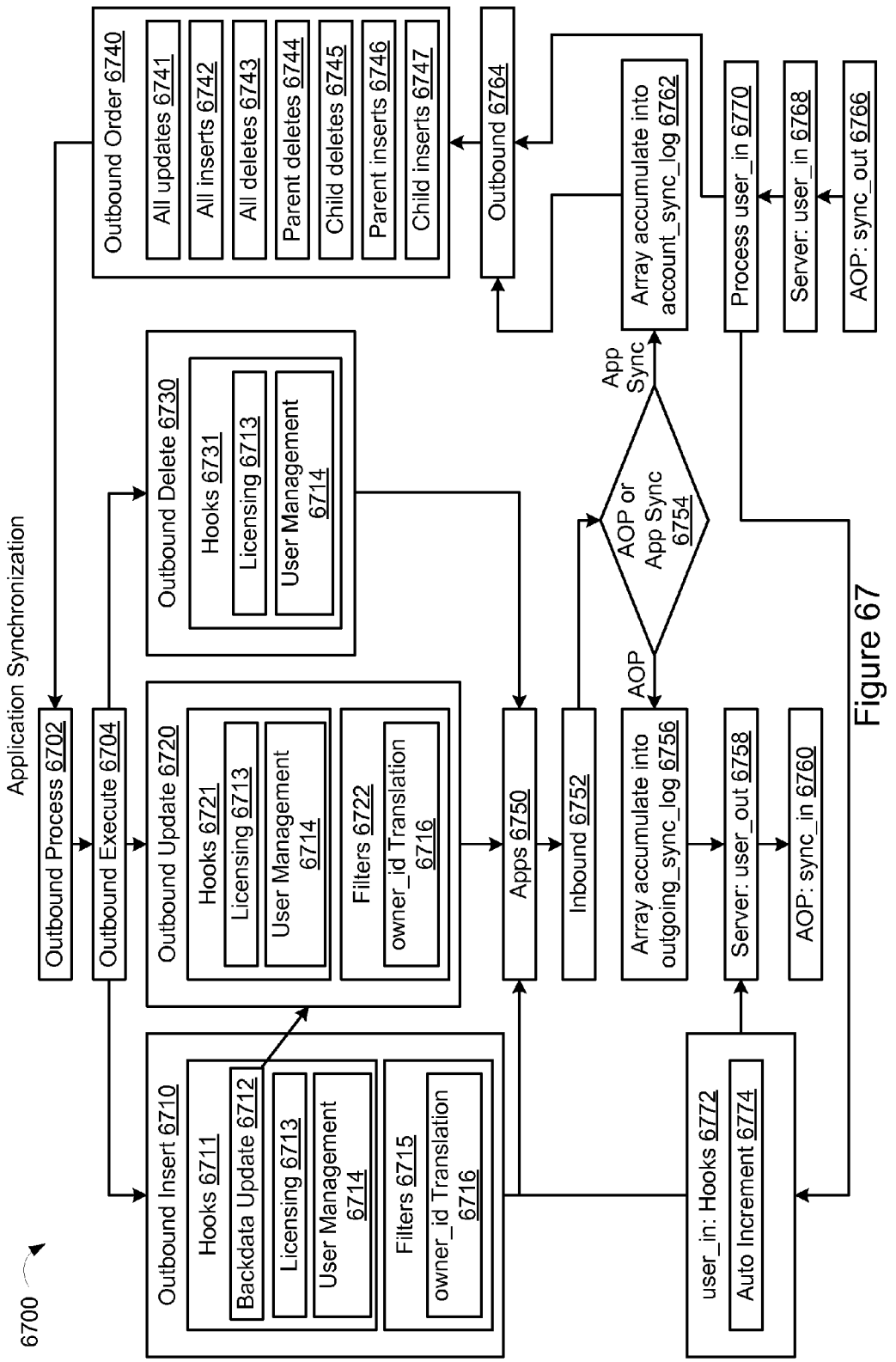

6700
Application Synchronization
Outbound Process 6702
Outbound Execute 6704
Outbound Insert 6710
Hooks 6711
Backdata Update 6712
Licensing 6713
User Management 6714
Filters 6715
owner_id Translation 6716
Outbound Update 6720
Hooks 6721
Licensing 6713
User Management 6714
Filters 6722
owner_id Translation 6716
Outbound Delete 6730
Hooks 6731
Licensing 6713
User Management 6714
Outbound Order 6740
All updates 6741
All inserts 6742
All deletes 6743
Parent deletes 6744
Child deletes 6745
Parent inserts 6746
Child inserts 6747
Apps 6750
Inbound 6752
AOP or App Sync 6754
AOP
App Sync
Array accumulate into outgoing_sync_log 6756
Server: user_out 6758
AOP: sync_in 6760
user_in: Hooks 6772
Auto Increment 6774
Outbound 6764
Array accumulate into account_sync_log 6762
Process user_in 6770
Server: user_in 6768
AOP: sync_out 6766

Figure 67

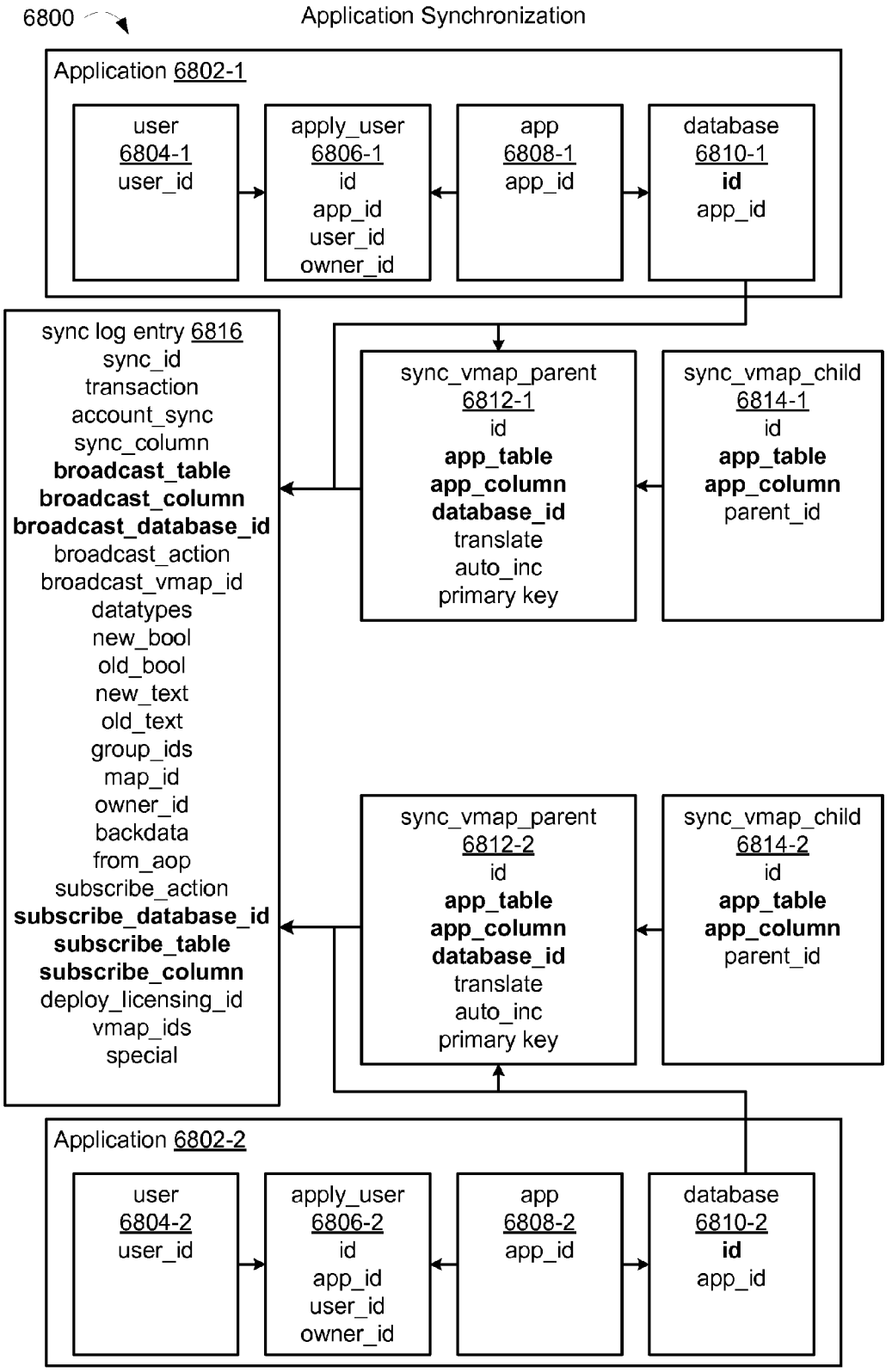

6800
Application Synchronization
Application 6802-1
user
6804-1
user_id
apply_user
6806-1
id
app_id
user_id
owner_id
app
6808-1
app_id
database
6810-1
id
app_id
sync log entry 6816
sync_id
transaction
account_sync
sync_column
broadcast_table
broadcast_column
broadcast_database_id
broadcast_action
broadcast_vmap_id
datatypes
new_bool
old_bool
new_text
old_text
group_ids
map_id
owner_id
backdata
from_aop
subscribe_action
subscribe_database_id
subscribe_table
subscribe_column
deploy_licensing_id
vmap_ids
special
sync_vmap_parent
6812-1
id
app_table
app_column
database_id
translate
auto_inc
primary key
sync_vmap_child
6814-1
id
app_table
app_column
parent_id
sync_vmap_parent
6812-2
id
app_table
app_column
database_id
translate
auto_inc
primary key
sync_vmap_child
6814-2
id
app_table
app_column
parent_id
Application 6802-2
user
6804-2
user_id
apply_user
6806-2
id
app_id
user_id
owner_id
app
6808-2
app_id
database
6810-2
id
app_id

Figure 68

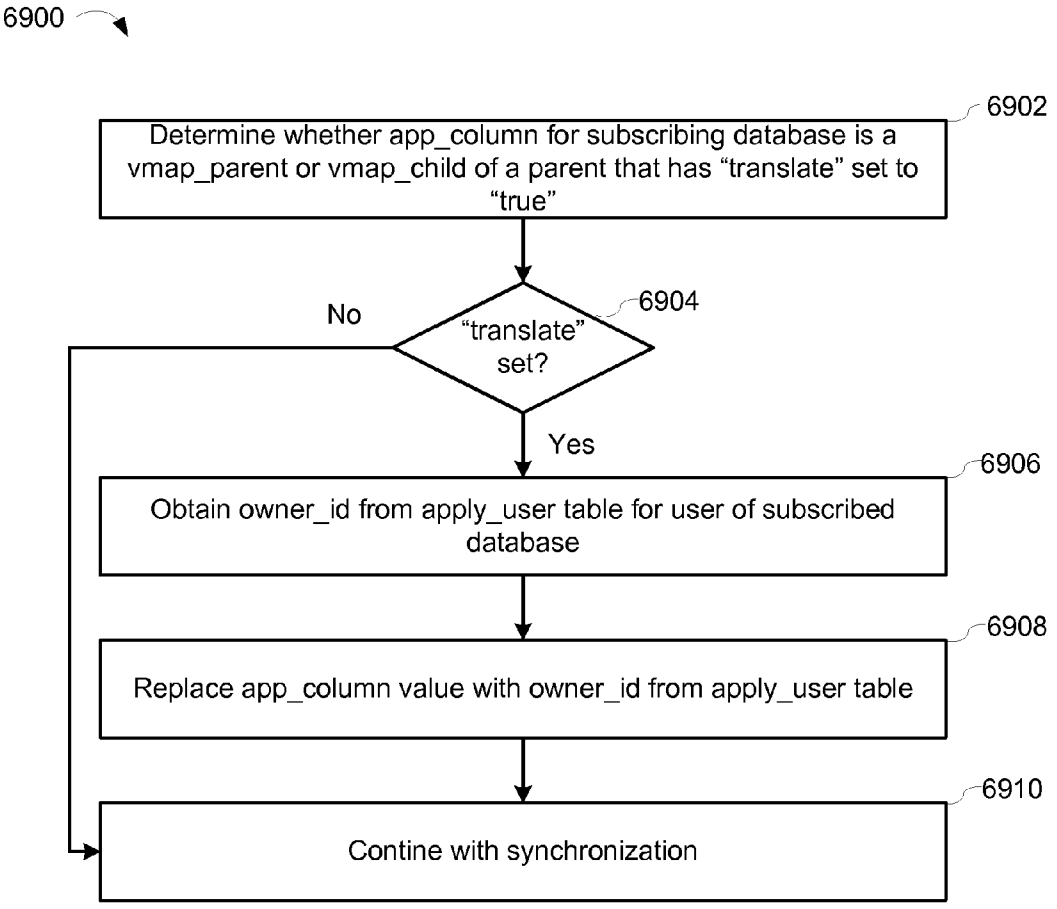

6900
6902
Determine whether app_column for subscribing database is a vmap_parent or vmap_child of a parent that has "translate" set to "true"
6904
"translate" set?
No
Yes
6906
Obtain owner_id from apply_user table for user of subscribed database
6908
Replace app_column value with owner_id from apply_user table
6910
Contine with synchronization

Figure 69

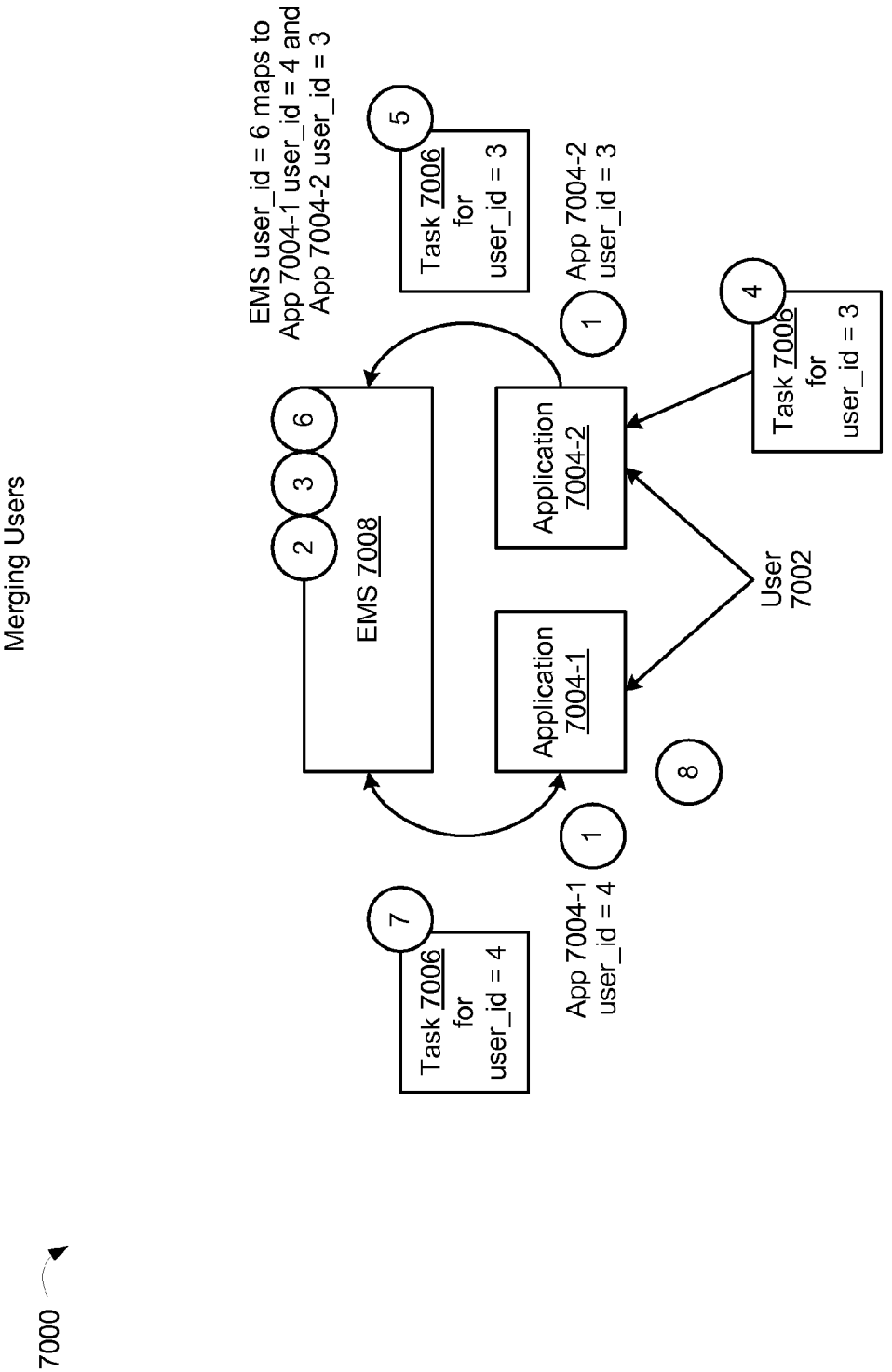

7000
Merging Users
EMS user_id = 6 maps to App 7004-1 user_id = 4 and App 7004-2 user_id = 3
EMS 7008
Application 7004-1
Application 7004-2
User 7002
App 7004-1 user_id = 4
App 7004-2 user_id = 3
Task 7006 for user_id = 4
Task 7006 for user_id = 3
Task 7006 for user_id = 3
1
2
3
4
5
6
7
8

Figure 70

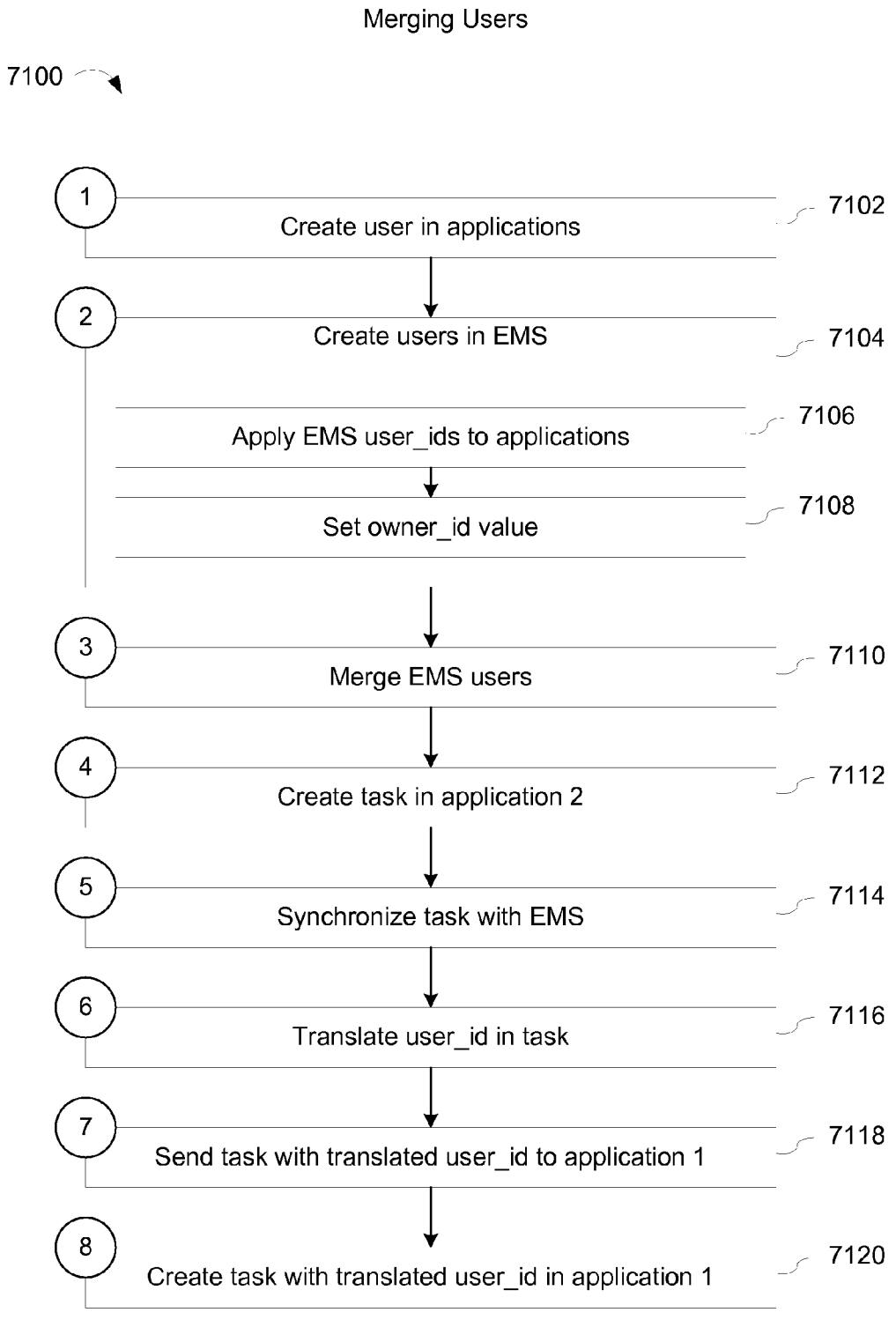

Merging Users
7100
1
Create user in applications
7102
2
Create users in EMS
7104
Apply EMS user_ids to applications
7106
Set owner_id value
7108
3
Merge EMS users
7110
4
Create task in application 2
7112
5
Synchronize task with EMS
7114
6
Translate user_id in task
7116
7
Send task with translated user_id to application 1
7118
8
Create task with translated user_id in application 1
7120

Figure 71

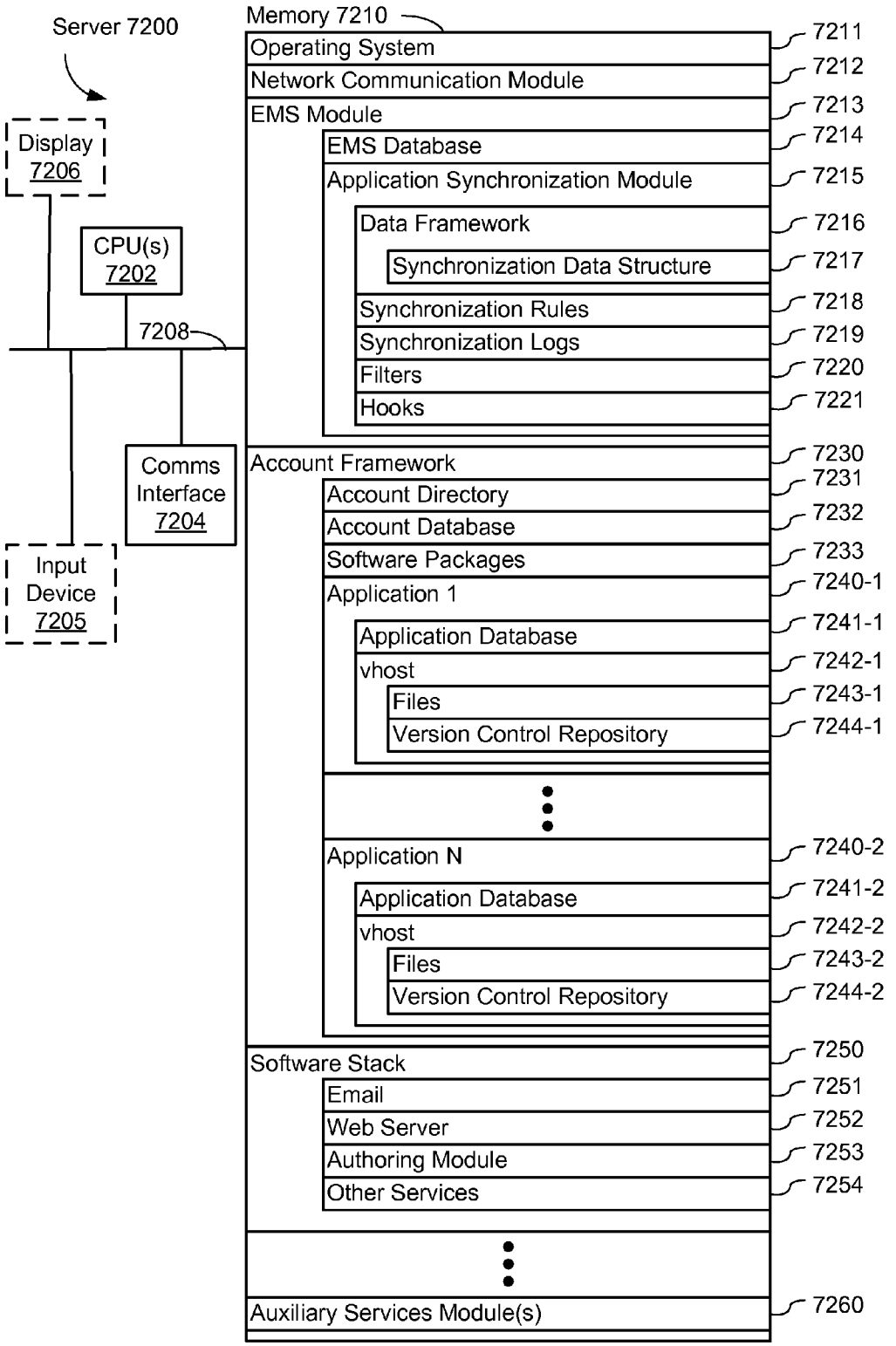

Server 7200
Display 7206
CPU(s) 7202
7208
Comms Interface 7204
Input Device 7205
Memory 7210
Operating System
7211
Network Communication Module
7212
EMS Module
7213
EMS Database
7214
Application Synchronization Module
7215
Data Framework
7216
Synchronization Data Structure
7217
Synchronization Rules
7218
Synchronization Logs
7219
Filters
7220
Hooks
7221
Account Framework
7230
Account Directory
7231
Account Database
7232
Software Packages
7233
Application 1
7240-1
Application Database
7241-1
vhost
7242-1
Files
7243-1
Version Control Repository
7244-1
Application N
7240-2
Application Database
7241-2
vhost
7242-2
Files
7243-2
Version Control Repository
7244-2
Software Stack
7250
Email
7251
Web Server
7252
Authoring Module
7253
Other Services
7254
Auxiliary Services Module(s)
7260

Figure 72

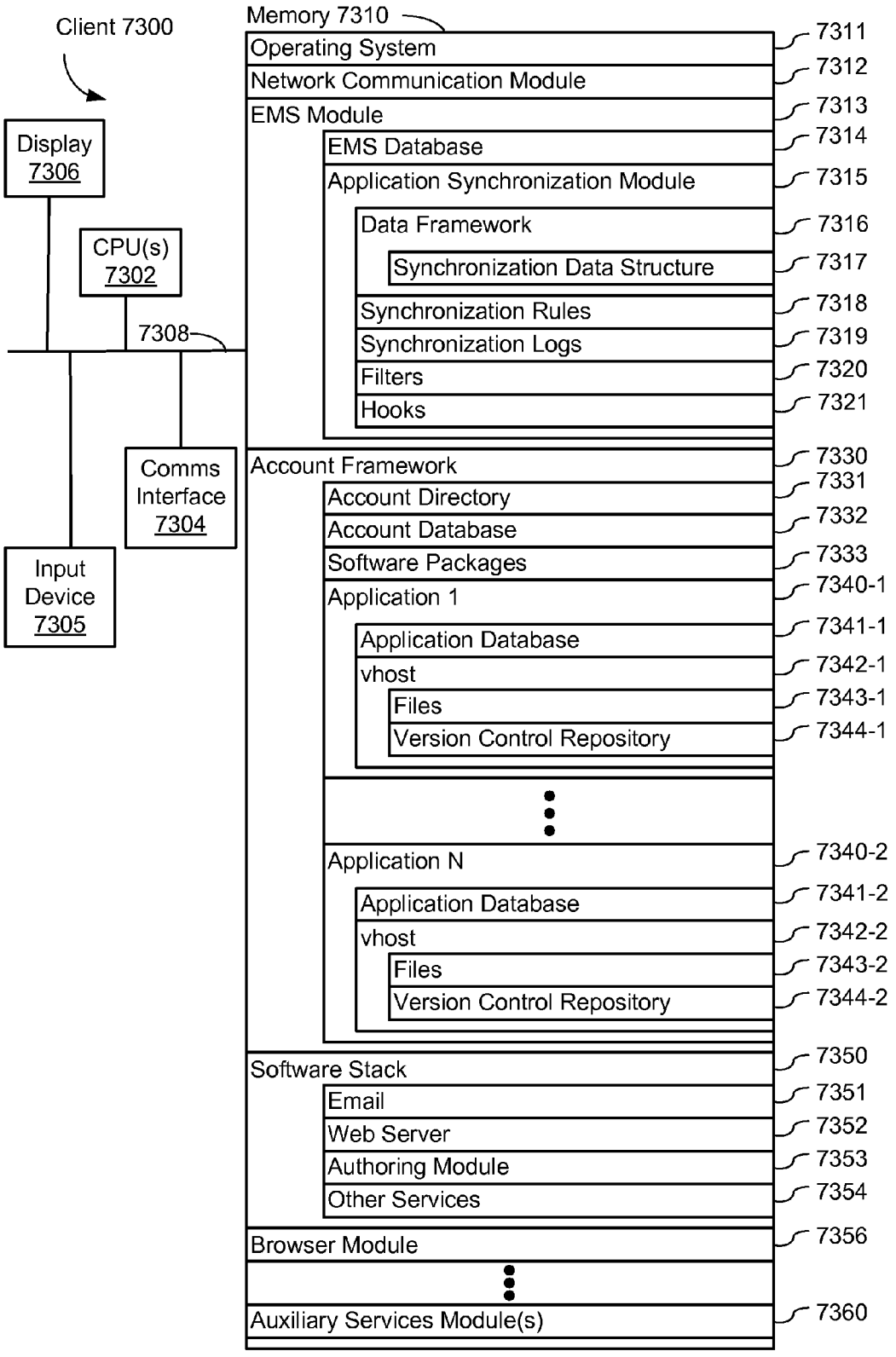

Client 7300
Display 7306
CPU(s) 7302
7308
Comms Interface 7304
Input Device 7305
Memory 7310
Operating System
7311
Network Communication Module
7312
EMS Module
7313
EMS Database
7314
Application Synchronization Module
7315
Data Framework
7316
Synchronization Data Structure
7317
Synchronization Rules
7318
Synchronization Logs
7319
Filters
7320
Hooks
7321
Account Framework
7330
Account Directory
7331
Account Database
7332
Software Packages
7333
Application 1
7340-1
Application Database
7341-1
vhost
7342-1
Files
7343-1
Version Control Repository
7344-1
Application N
7340-2
Application Database
7341-2
vhost
7342-2
Files
7343-2
Version Control Repository
7344-2
Software Stack
7350
Email
7351
Web Server
7352
Authoring Module
7353
Other Services
7354
Browser Module
7356
Auxiliary Services Module(s)
7360

Figure 73

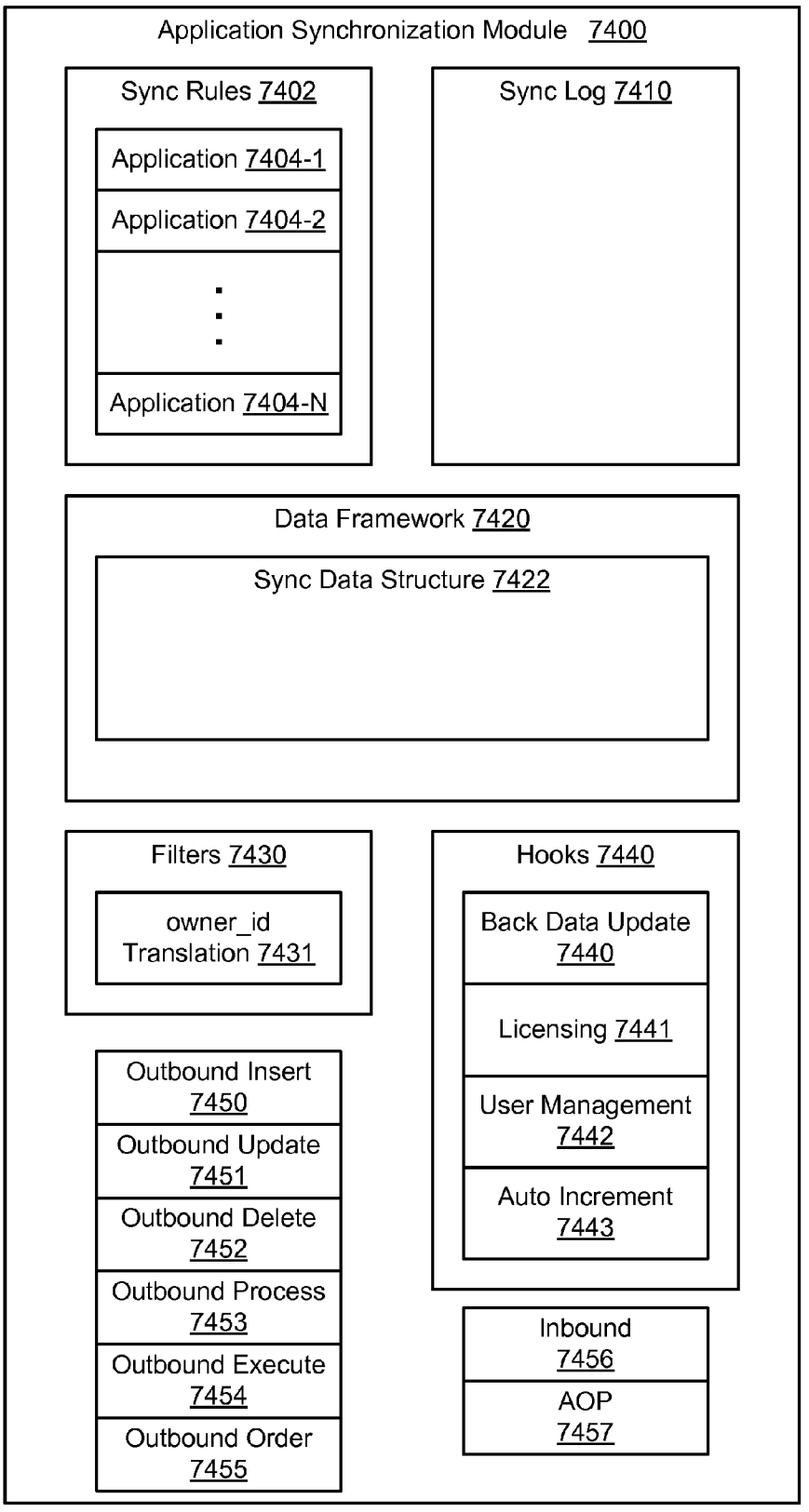

Application Synchronization Module 7400
Sync Rules 7402
Application 7404-1
Application 7404-2
Application 7404-N
Sync Log 7410
Data Framework 7420
Sync Data Structure 7422
Filters 7430
owner_id Translation 7431
Hooks 7440
Back Data Update 7440
Licensing 7441
User Management 7442
Auto Increment 7443
Outbound Insert 7450
Outbound Update 7451
Outbound Delete 7452
Outbound Process 7453
Outbound Execute 7454
Outbound Order 7455
Inbound 7456
AOP 7457

Figure 74

7500
eas_sync_app_log 7502
created_on
id
transaction
account_sync
backdata
publish
vmap_id
map_id
action
datatype
app_table
app_column
sync_table
sync_column
old_bool
new_bool
old_text
new_text
group_ids
owner_id
get_children
inbound 7504
created_on
sync_id
transaction
account_sync
database_id
backdata
map_id
action
datatype
app_table
app_column
sync_table
sync_column
old_bool
new_bool
old_text
new_text
app_created_on
backdata_database
groups
group_ids
owner_id
get_children
from_aop
broadcast_vmap_id
outbound 7506
sync_id
transaction
account_sync
sync_table
sync_column
broadcast_table
broadcast_column
broadcast_database_id
broadcast_action
broadcast_vmap_id
datatypes
new_bool
old_bool
new_text
old_text
group_ids
map_id
owner_id
backdata
from_aop
subscribe_action
subscribe_database_id
subscribe_table
subscribe_column
deploy_licensing_id
vmap_ids
preserve_id
special
created_on

Figure 75

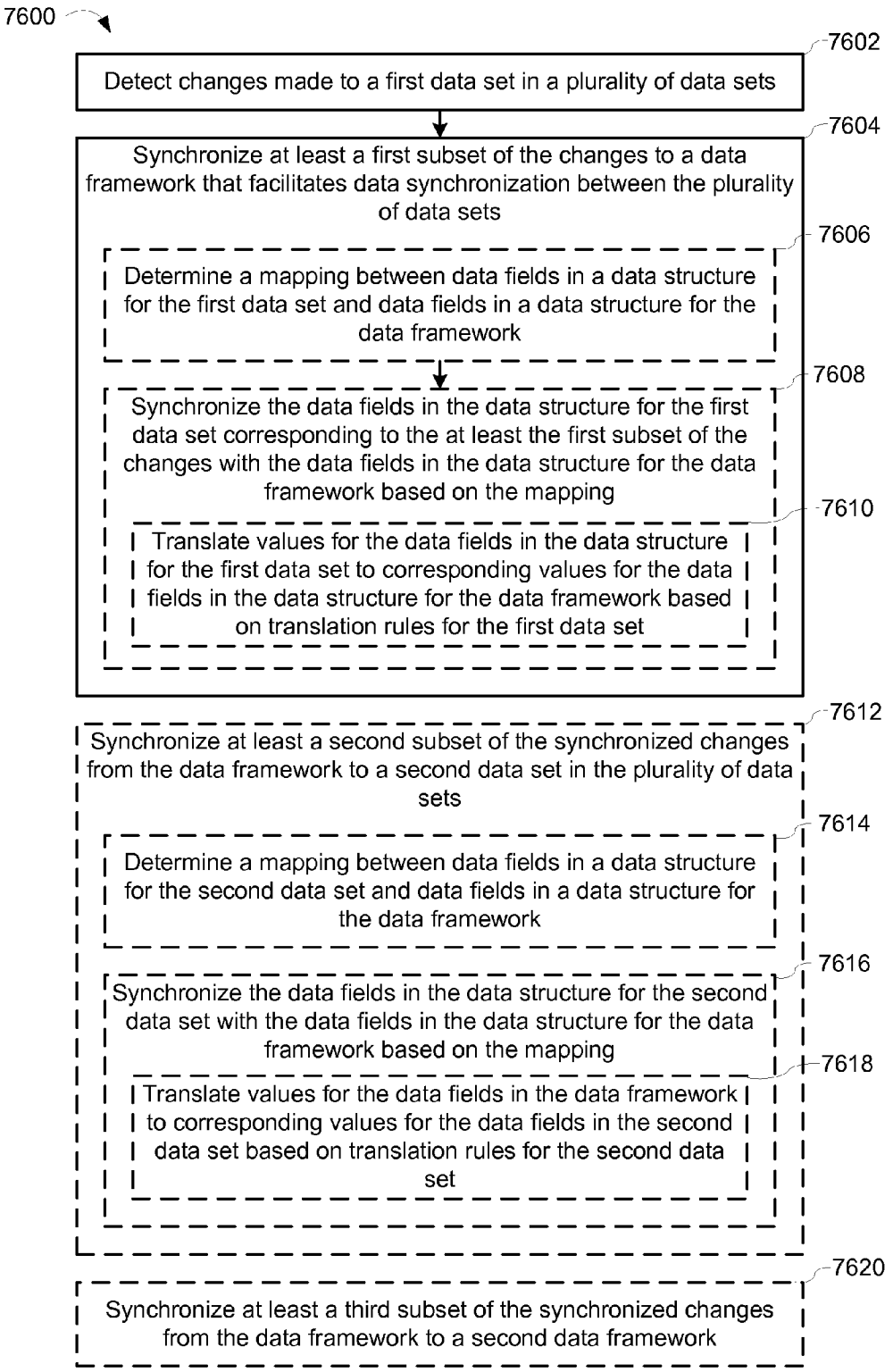

7600
7602
Detect changes made to a first data set in a plurality of data sets
7604
Synchronize at least a first subset of the changes to a data framework that facilitates data synchronization between the plurality of data sets
7606
Determine a mapping between data fields in a data structure for the first data set and data fields in a data structure for the data framework
7608
Synchronize the data fields in the data structure for the first data set corresponding to the at least the first subset of the changes with the data fields in the data structure for the data framework based on the mapping
7610
Translate values for the data fields in the data structure for the first data set to corresponding values for the data fields in the data structure for the data framework based on translation rules for the first data set
7612
Synchronize at least a second subset of the synchronized changes from the data framework to a second data set in the plurality of data sets
7614
Determine a mapping between data fields in a data structure for the second data set and data fields in a data structure for the data framework
7616
Synchronize the data fields in the data structure for the second data set with the data fields in the data structure for the data framework based on the mapping
7618
Translate values for the data fields in the data framework to corresponding values for the data fields in the second data set based on translation rules for the second data set
7620
Synchronize at least a third subset of the synchronized changes from the data framework to a second data framework

Figure 76

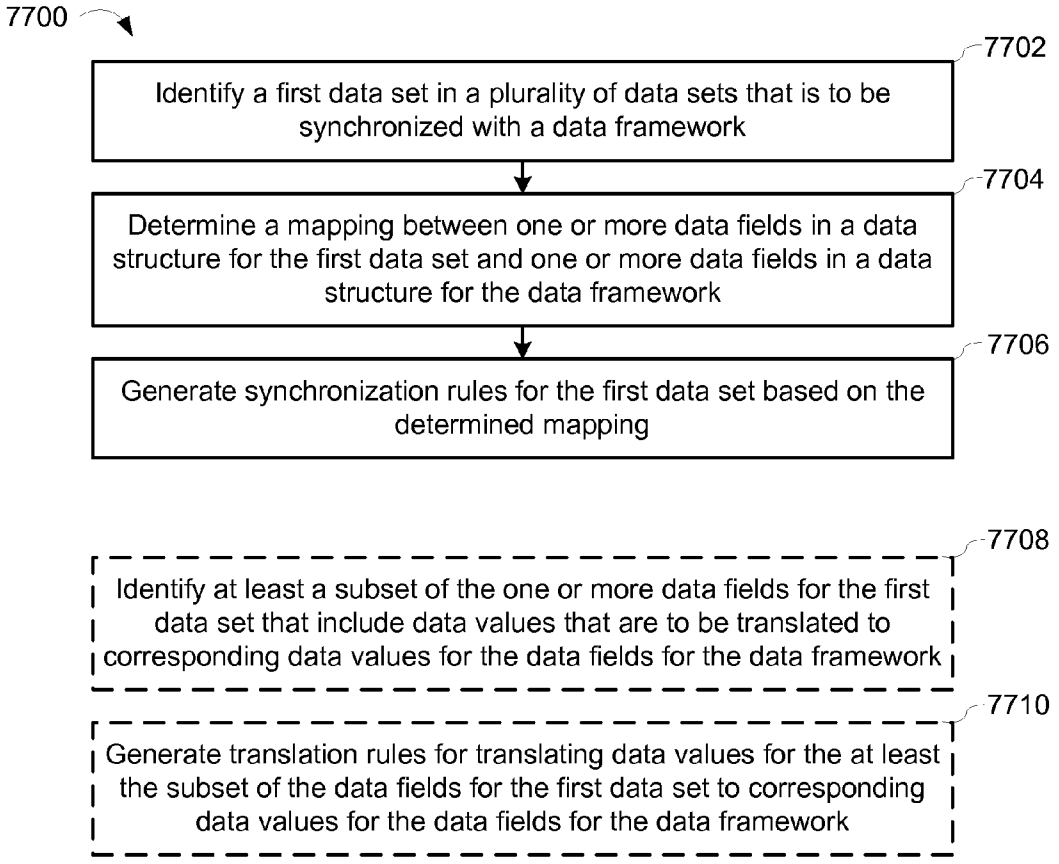

7700
7702
Identify a first data set in a plurality of data sets that is to be synchronized with a data framework
7704
Determine a mapping between one or more data fields in a data structure for the first data set and one or more data fields in a data structure for the data framework
7706
Generate synchronization rules for the first data set based on the determined mapping
7708
Identify at least a subset of the one or more data fields for the first data set that include data values that are to be translated to corresponding data values for the data fields for the data framework
7710
Generate translation rules for translating data values for the at least the subset of the data fields for the first data set to corresponding data values for the data fields for the data framework

Figure 77

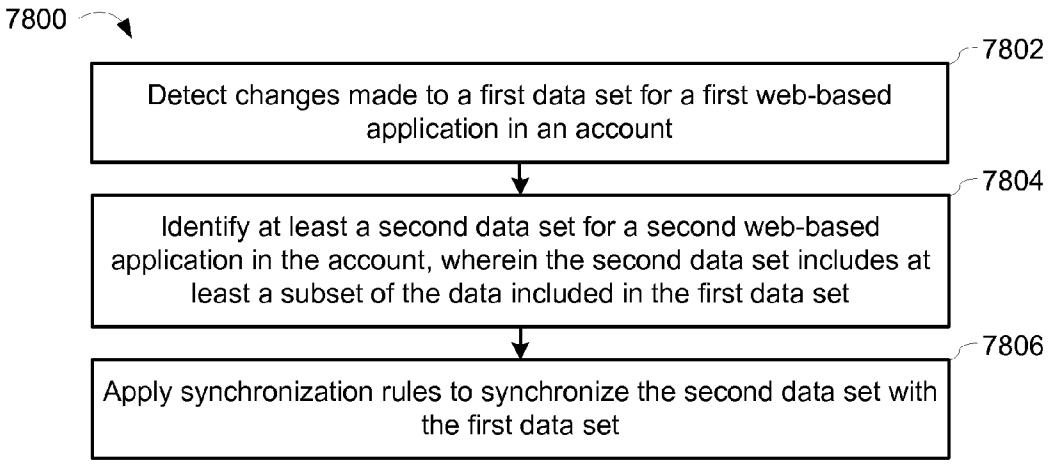

7800
7802
Detect changes made to a first data set for a first web-based application in an account
7804
Identify at least a second data set for a second web-based application in the account, wherein the second data set includes at least a subset of the data included in the first data set
7806
Apply synchronization rules to synchronize the second data set with the first data set

Figure 78

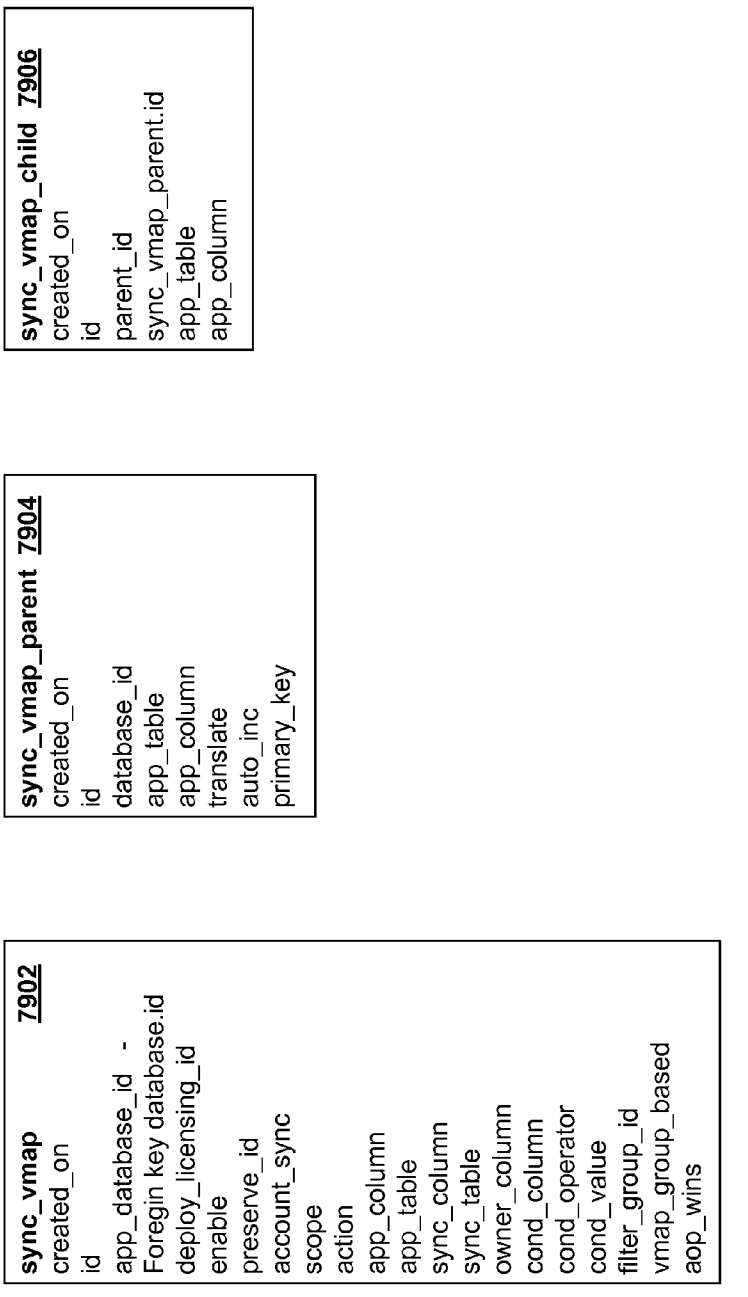

7900
sync_vmap 7902
created_on
id
app_database_id -
Foregin key database.id
deploy_licensing_id
enable
preserve_id
account_sync
scope
action
app_column
app_table
sync_column
sync_table
owner_column
cond_column
cond_operator
cond_value
filter_group_id
vmap_group_based
aop_wins
sync_vmap_parent 7904
created_on
id
database_id
app_table
app_column
translate
auto_inc
primary_key
sync_vmap_child 7906
created_on
id
parent_id
sync_vmap_parent.id
app_table
app_column

Figure 79

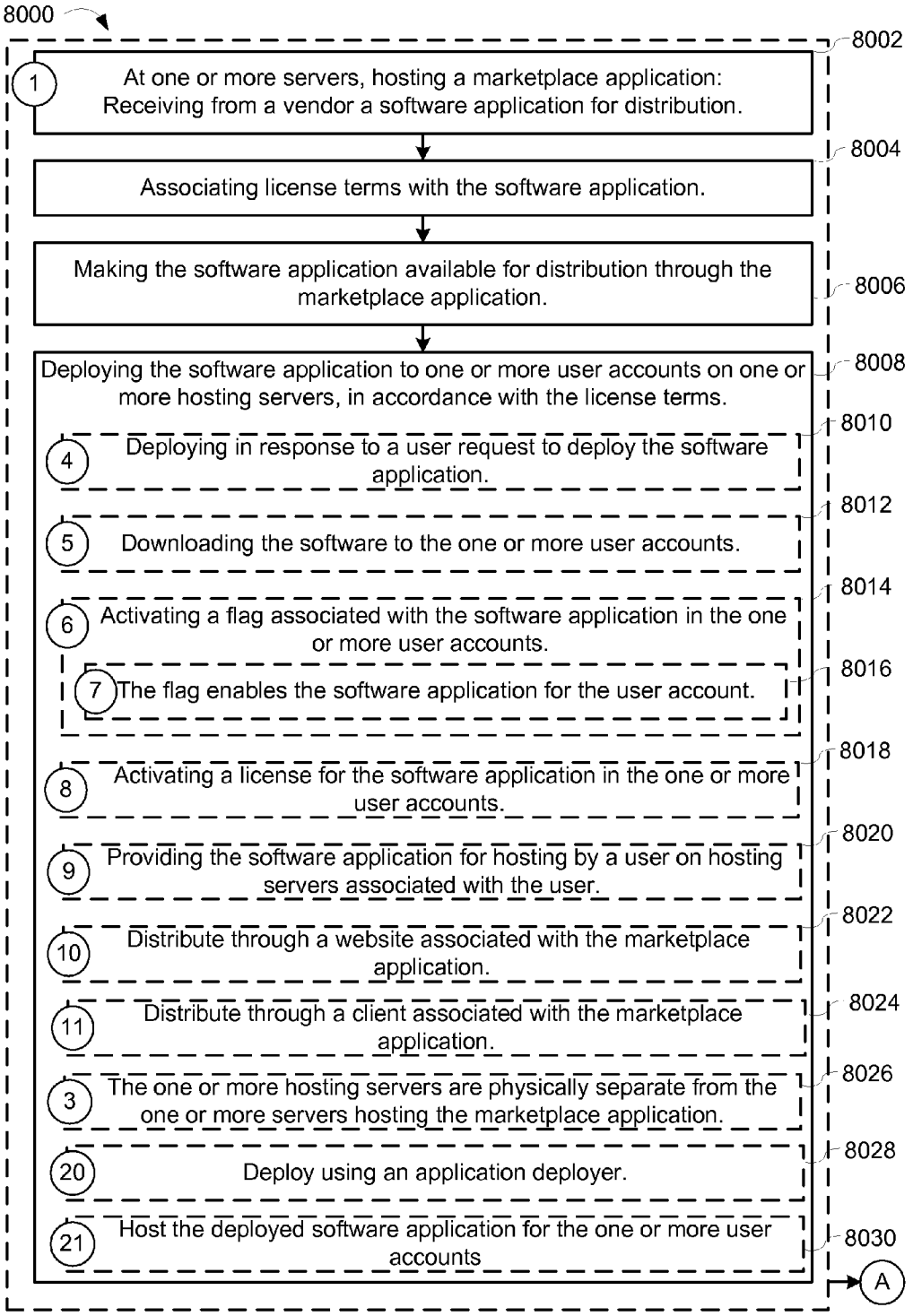

8000
8002
1
At one or more servers, hosting a marketplace application:
Receiving from a vendor a software application for distribution.
8004
Associating license terms with the software application.
Making the software application available for distribution through the marketplace application.
8006
Deploying the software application to one or more user accounts on one or more hosting servers, in accordance with the license terms.
8008
8010
4
Deploying in response to a user request to deploy the software application.
8012
5
Downloading the software to the one or more user accounts.
8014
6
Activating a flag associated with the software application in the one or more user accounts.
8016
7
The flag enables the software application for the user account.
8018
8
Activating a license for the software application in the one or more user accounts.
8020
9
Providing the software application for hosting by a user on hosting servers associated with the user.
8022
10
Distribute through a website associated with the marketplace application.
8024
11
Distribute through a client associated with the marketplace application.
8026
3
The one or more hosting servers are physically separate from the one or more servers hosting the marketplace application.
8028
20
Deploy using an application deployer.
8030
21
Host the deployed software application for the one or more user accounts
A

Figure 80

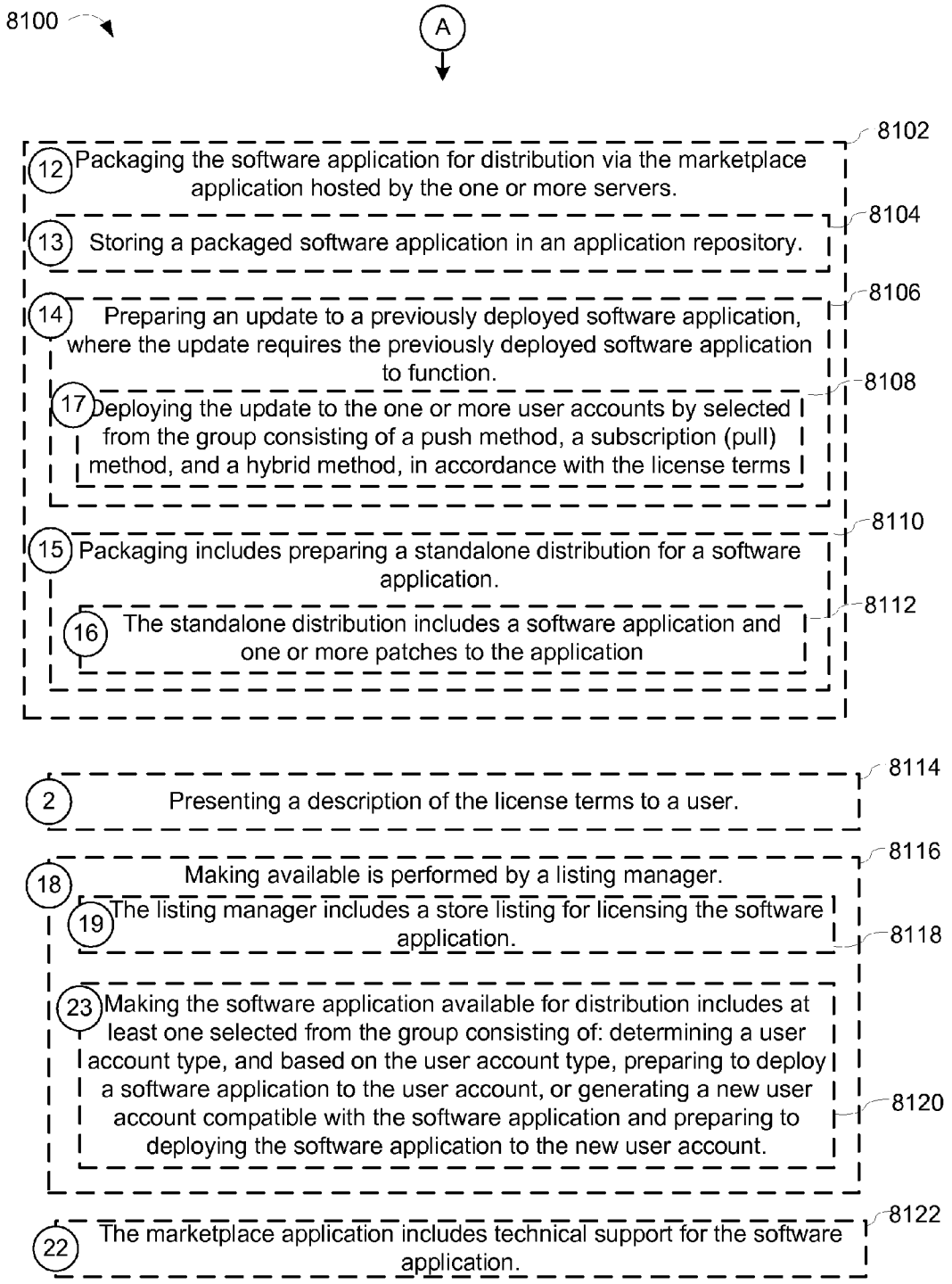

8100
A
8102
12 Packaging the software application for distribution via the marketplace application hosted by the one or more servers.
8104
13 Storing a packaged software application in an application repository.
8106
14 Preparing an update to a previously deployed software application, where the update requires the previously deployed software application to function.
8108
17 Deploying the update to the one or more user accounts by selected from the group consisting of a push method, a subscription (pull) method, and a hybrid method, in accordance with the license terms
8110
15 Packaging includes preparing a standalone distribution for a software application.
8112
16 The standalone distribution includes a software application and one or more patches to the application
8114
2 Presenting a description of the license terms to a user.
8116
18 Making available is performed by a listing manager.
19 The listing manager includes a store listing for licensing the software application.
8118
23 Making the software application available for distribution includes at least one selected from the group consisting of: determining a user account type, and based on the user account type, preparing to deploy a software application to the user account, or generating a new user account compatible with the software application and preparing to deploying the software application to the new user account.
8120
8122
22 The marketplace application includes technical support for the software application.

Figure 81

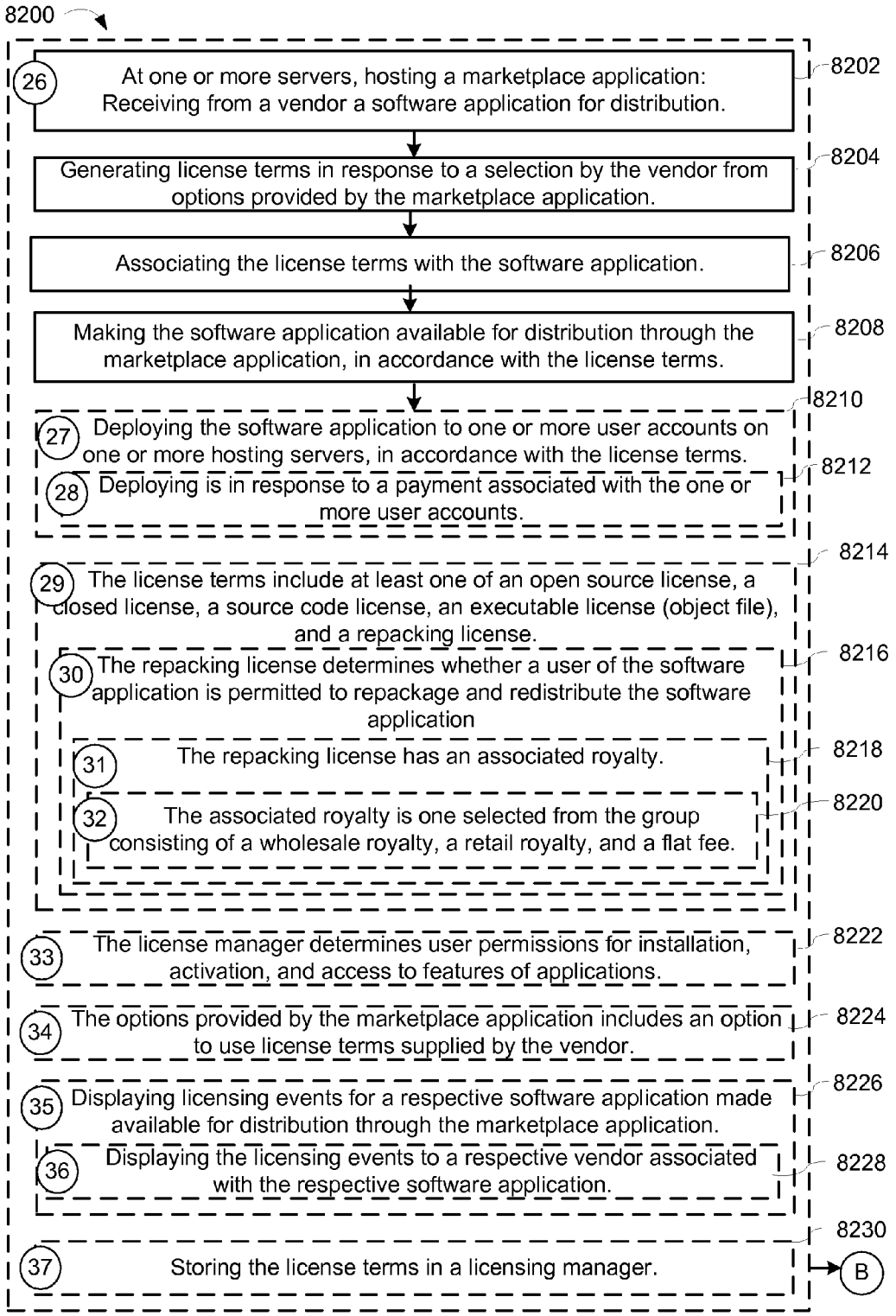

8200
26
At one or more servers, hosting a marketplace application: Receiving from a vendor a software application for distribution.
8202
Generating license terms in response to a selection by the vendor from options provided by the marketplace application.
8204
Associating the license terms with the software application.
8206
Making the software application available for distribution through the marketplace application, in accordance with the license terms.
8208
8210
27
Deploying the software application to one or more user accounts on one or more hosting servers, in accordance with the license terms.
8212
28
Deploying is in response to a payment associated with the one or more user accounts.
8214
29
The license terms include at least one of an open source license, a closed license, a source code license, an executable license (object file), and a repacking license.
8216
30
The repacking license determines whether a user of the software application is permitted to repackage and redistribute the software application
8218
31
The repacking license has an associated royalty.
8220
32
The associated royalty is one selected from the group consisting of a wholesale royalty, a retail royalty, and a flat fee.
8222
33
The license manager determines user permissions for installation, activation, and access to features of applications.
8224
34
The options provided by the marketplace application includes an option to use license terms supplied by the vendor.
8226
35
Displaying licensing events for a respective software application made available for distribution through the marketplace application.
8228
36
Displaying the licensing events to a respective vendor associated with the respective software application.
8230
37
Storing the license terms in a licensing manager.
B

Figure 82

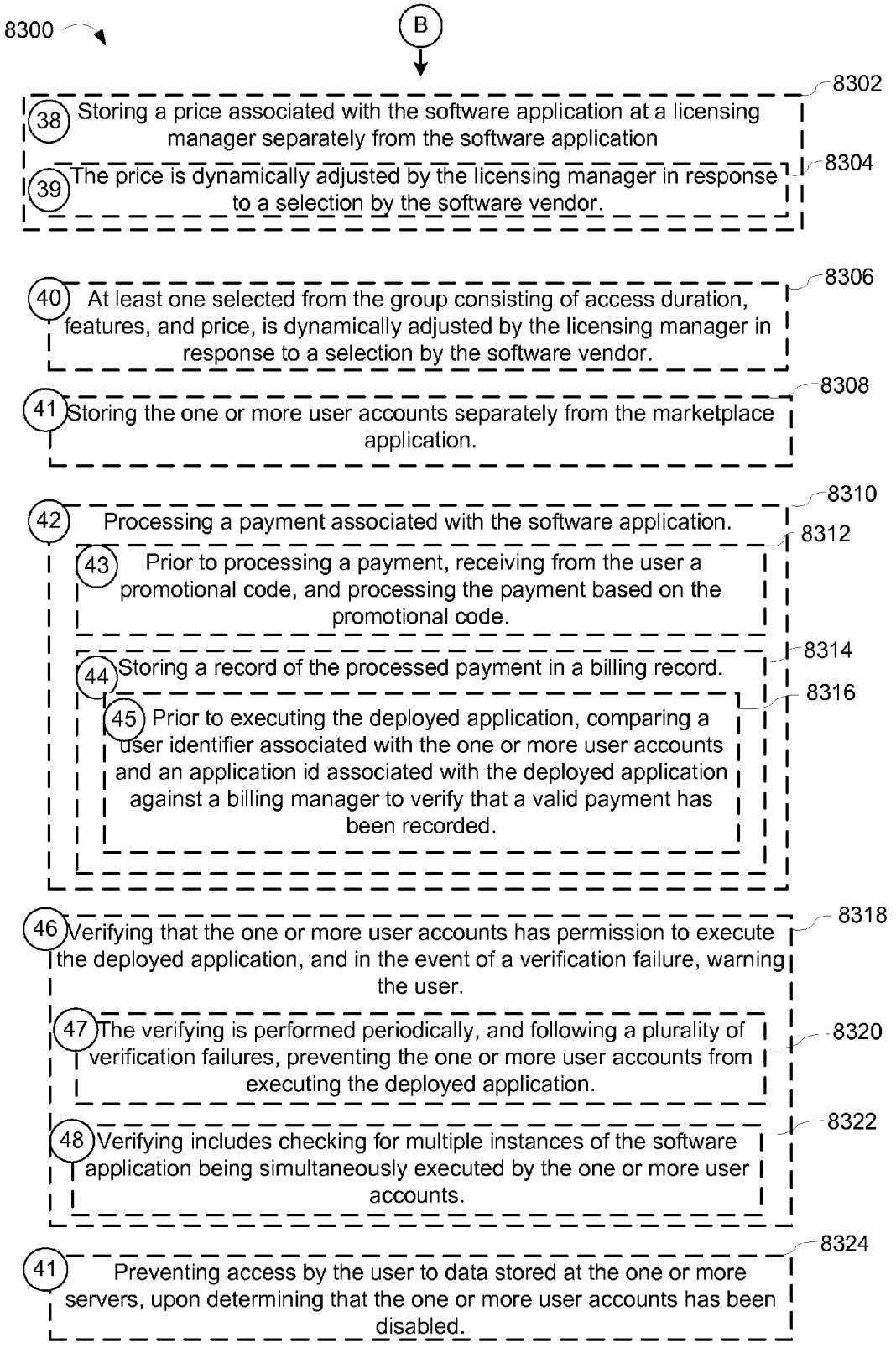

8300
B
8302
38 Storing a price associated with the software application at a licensing manager separately from the software application
8304
39 The price is dynamically adjusted by the licensing manager in response to a selection by the software vendor.
8306
40 At least one selected from the group consisting of access duration, features, and price, is dynamically adjusted by the licensing manager in response to a selection by the software vendor.
8308
41 Storing the one or more user accounts separately from the marketplace application.
8310
42 Processing a payment associated with the software application.
8312
43 Prior to processing a payment, receiving from the user a promotional code, and processing the payment based on the promotional code.
8314
44 Storing a record of the processed payment in a billing record.
8316
45 Prior to executing the deployed application, comparing a user identifier associated with the one or more user accounts and an application id associated with the deployed application against a billing manager to verify that a valid payment has been recorded.
8318
46 Verifying that the one or more user accounts has permission to execute the deployed application, and in the event of a verification failure, warning the user.
8320
47 The verifying is performed periodically, and following a plurality of verification failures, preventing the one or more user accounts from executing the deployed application.
8322
48 Verifying includes checking for multiple instances of the software application being simultaneously executed by the one or more user accounts.
8324
41 Preventing access by the user to data stored at the one or more servers, upon determining that the one or more user accounts has been disabled.

Figure 83

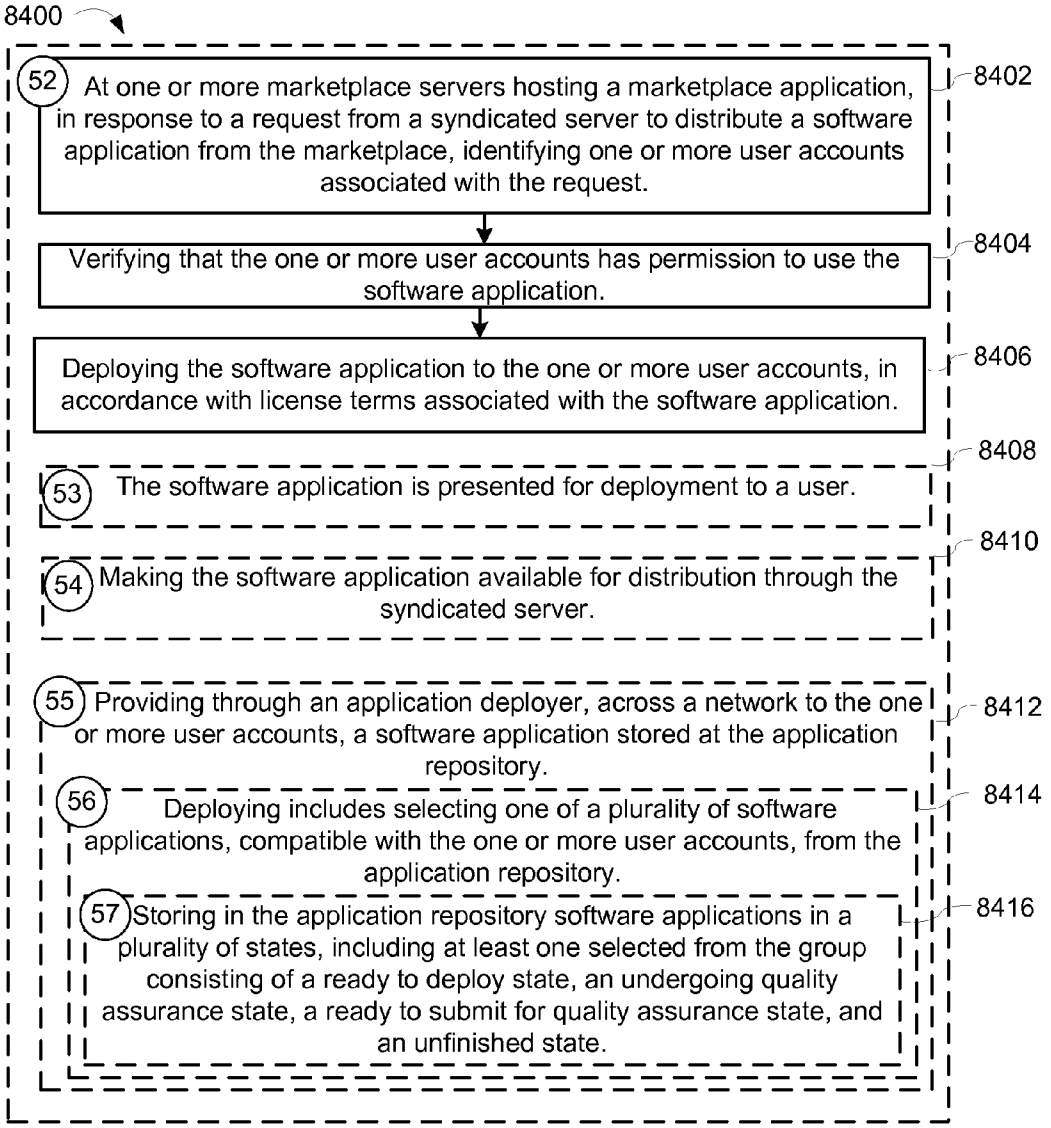

8400
52
At one or more marketplace servers hosting a marketplace application, in response to a request from a syndicated server to distribute a software application from the marketplace, identifying one or more user accounts associated with the request.
8402
Verifying that the one or more user accounts has permission to use the software application.
8404
Deploying the software application to the one or more user accounts, in accordance with license terms associated with the software application.
8406
53
The software application is presented for deployment to a user.
8408
54
Making the software application available for distribution through the syndicated server.
8410
55
Providing through an application deployer, across a network to the one or more user accounts, a software application stored at the application repository.
8412
56
Deploying includes selecting one of a plurality of software applications, compatible with the one or more user accounts, from the application repository.
8414
57
Storing in the application repository software applications in a plurality of states, including at least one selected from the group consisting of a ready to deploy state, an undergoing quality assurance state, a ready to submit for quality assurance state, and an unfinished state.
8416

Figure 84

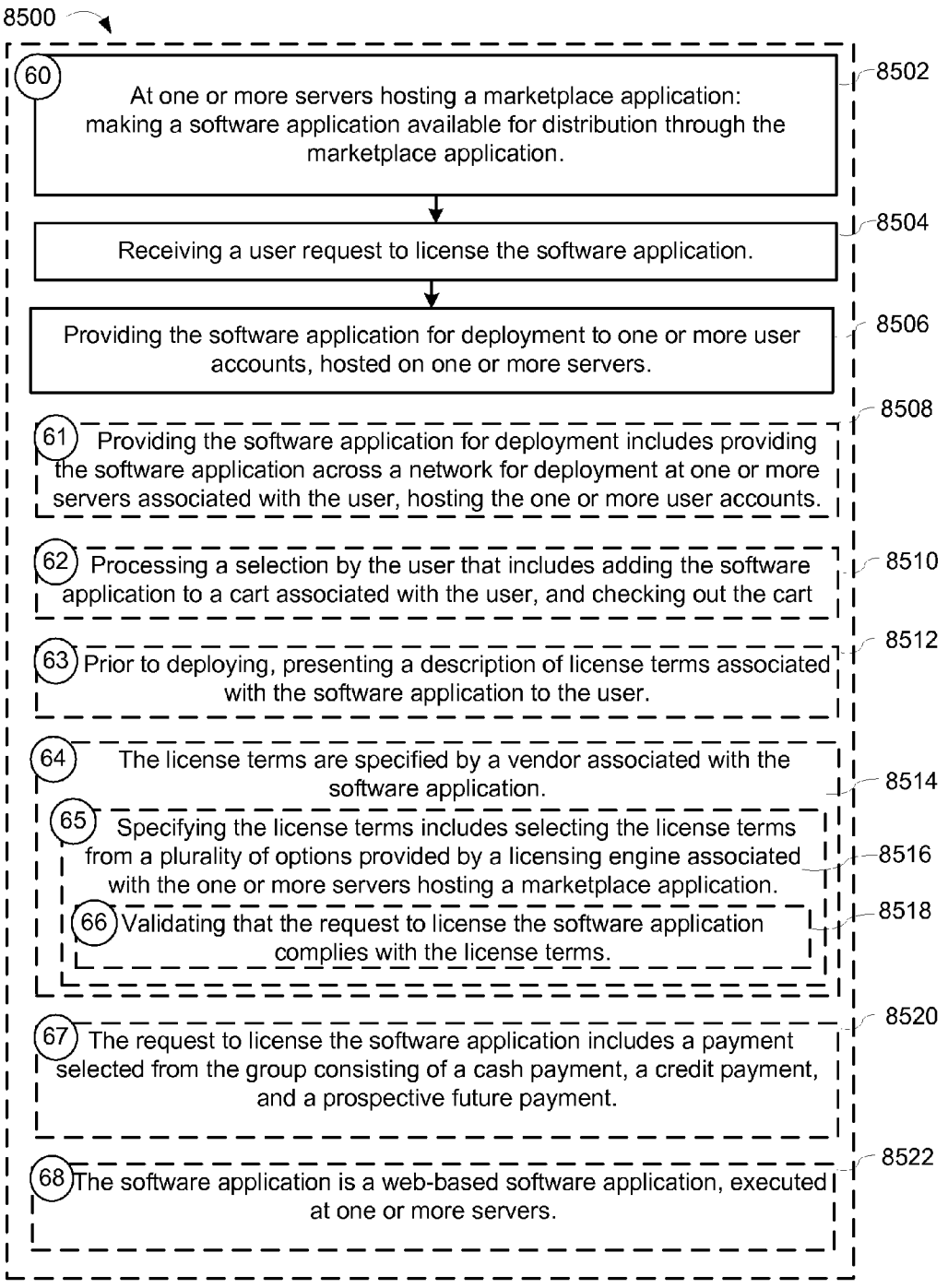

8500
60
At one or more servers hosting a marketplace application: making a software application available for distribution through the marketplace application.
8502
Receiving a user request to license the software application.
8504
Providing the software application for deployment to one or more user accounts, hosted on one or more servers.
8506
61
Providing the software application for deployment includes providing the software application across a network for deployment at one or more servers associated with the user, hosting the one or more user accounts.
8508
62
Processing a selection by the user that includes adding the software application to a cart associated with the user, and checking out the cart
8510
63
Prior to deploying, presenting a description of license terms associated with the software application to the user.
8512
64
The license terms are specified by a vendor associated with the software application.
8514
65
Specifying the license terms includes selecting the license terms from a plurality of options provided by a licensing engine associated with the one or more servers hosting a marketplace application.
8516
66
Validating that the request to license the software application complies with the license terms.
8518
67
The request to license the software application includes a payment selected from the group consisting of a cash payment, a credit payment, and a prospective future payment.
8520
68
The software application is a web-based software application, executed at one or more servers.
8522

Figure 85

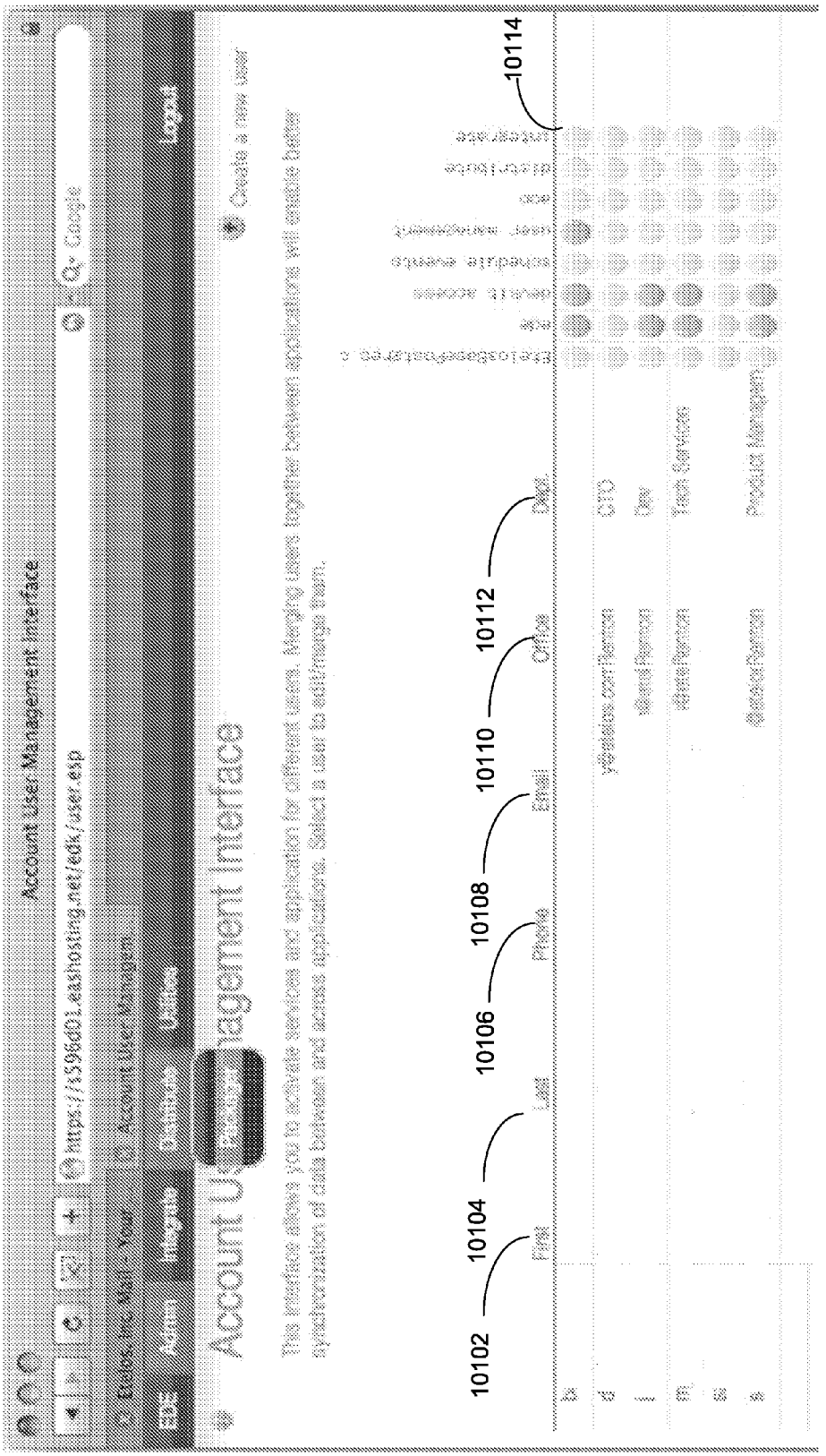

Account User Management Interface
Account User Management
Create a new user
10100
10102
10104
10106
10108
10110
10112
10114

Figure 101

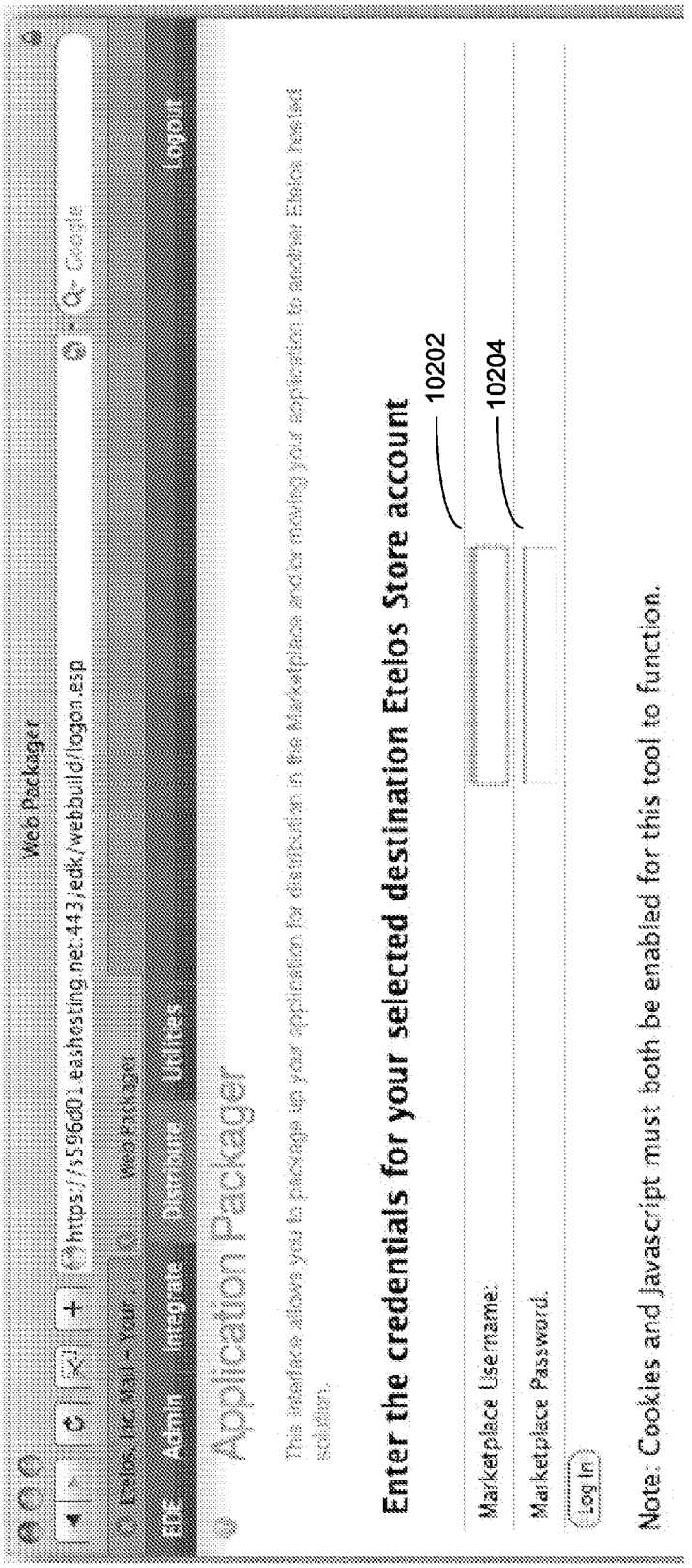

Web Packager
https://s596d01.eashosting.net:443/edk/webbuild/logon.esp
Google
Etelos, Inc. Mail - Your
Web Packager
EDF
Admin
Integrate
Distribute
Utilities
Logout
Application Packager
This interface allows you to package up your application for distribution in the Marketplace and/or moving your application to another Etelos hosted solution.
Enter the credentials for your selected destination Etelos Store account
10202
Marketplace Username:
10204
Marketplace Password:
Log In
Note: Cookies and Javascript must both be enabled for this tool to function.
10200

Figure 102

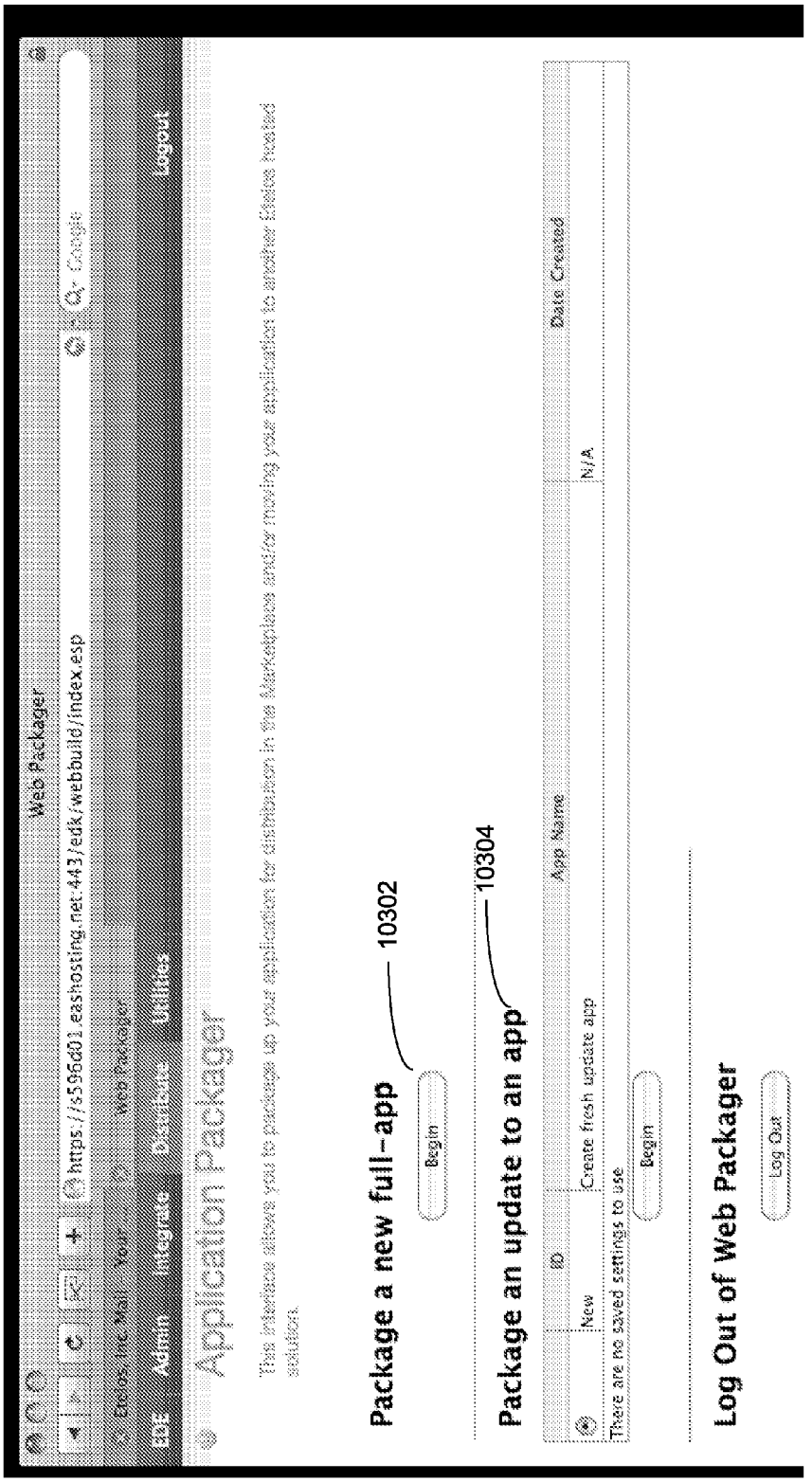

Web Packager
https://s596d01.eashosting.net:443/edk/webbuild/index.esp
Google
Etelos, Inc. Mail
Your
Web Packager
EDE
Admin
Integrate
Distribute
Utilities
Logout
Application Packager
This interface allows you to package up your application for distribution in the Marketplace and/or moving your application to another Etelos hosted solution.
Package a new full-app
Begin
10302
Package an update to an app
10304
ID
App Name
Date Created
New
Create fresh update app
N/A
There are no saved settings to use
Begin
Log Out of Web Packager
Log Out
10300

Figure 103

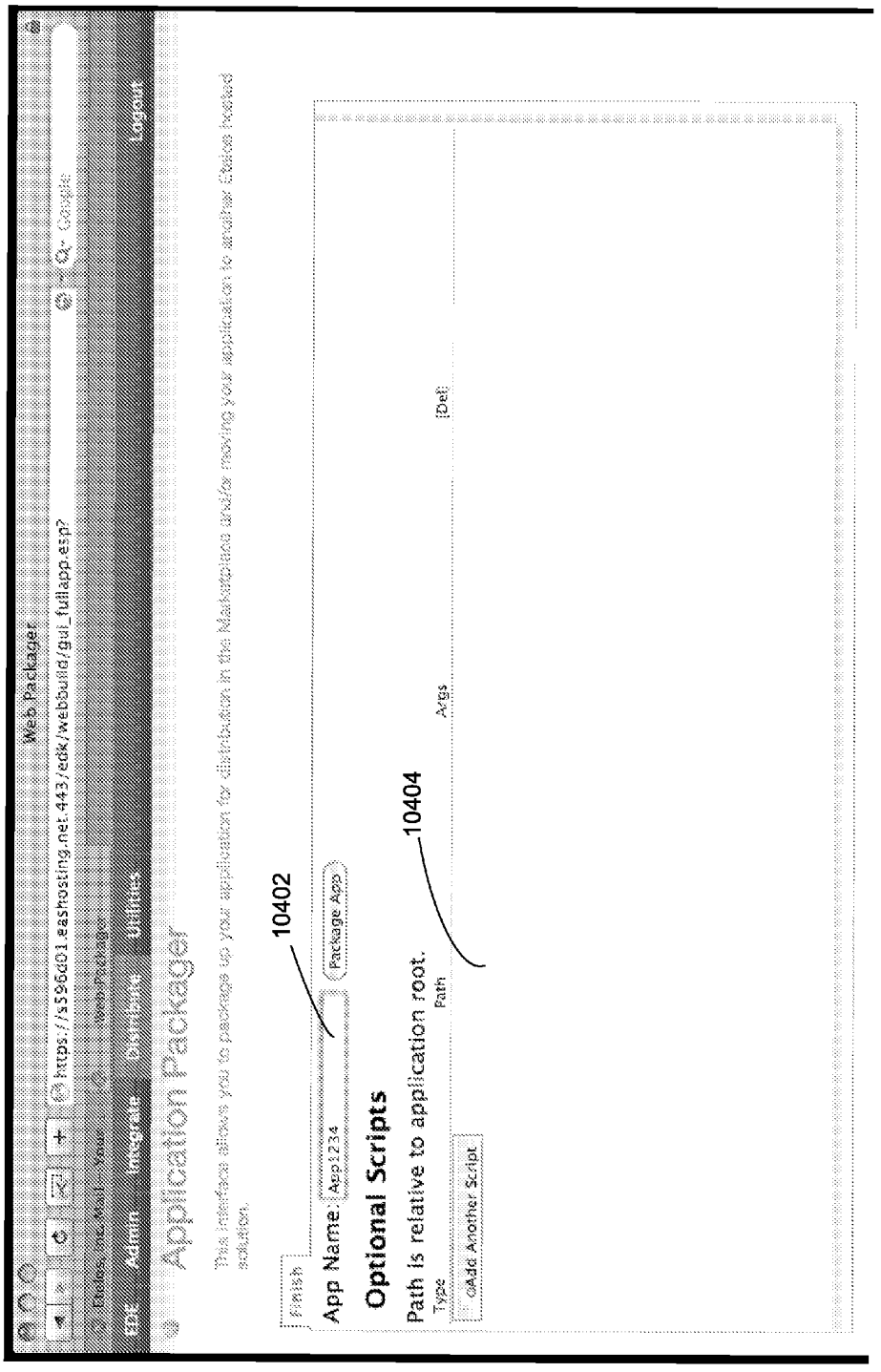

Web Packager
https://s596d01.sashosting.net:443/edk/webbuild/gui_fullapp.esp?
Application Packager
Finish
App Name: App1234
Package App
Optional Scripts
Path is relative to application root.
Type
Path
Args
[Del]
Add Another Script
10402
10404
10400

Figure 104

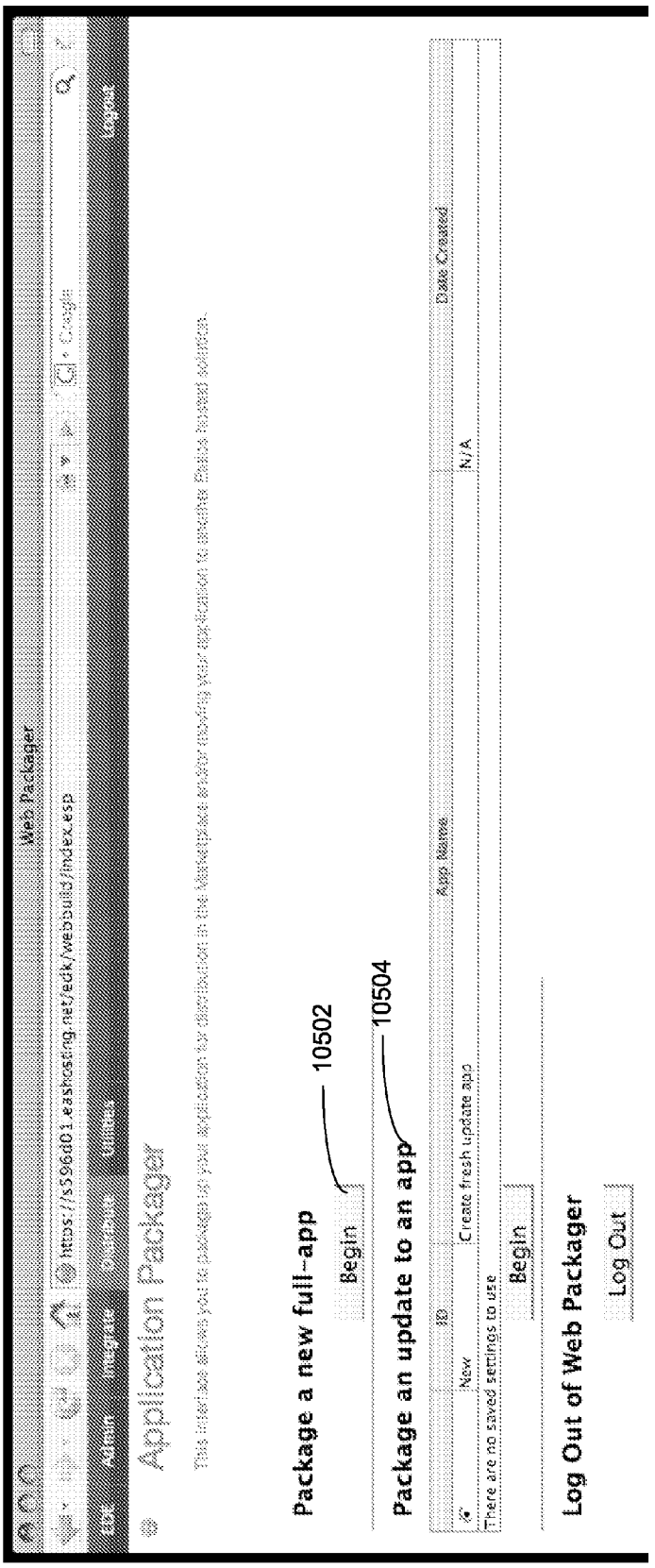

Web Packager
https://s596d01.eashosting.net/edk/webbuild/index.esp
Application Packager
Package a new full-app
Begin
10502
Package an update to an app
10504
ID
App Name
Date Created
New
Create fresh update app
N/A
There are no saved settings to use
Begin
Log Out of Web Packager
Log Out
10500

Figure 105

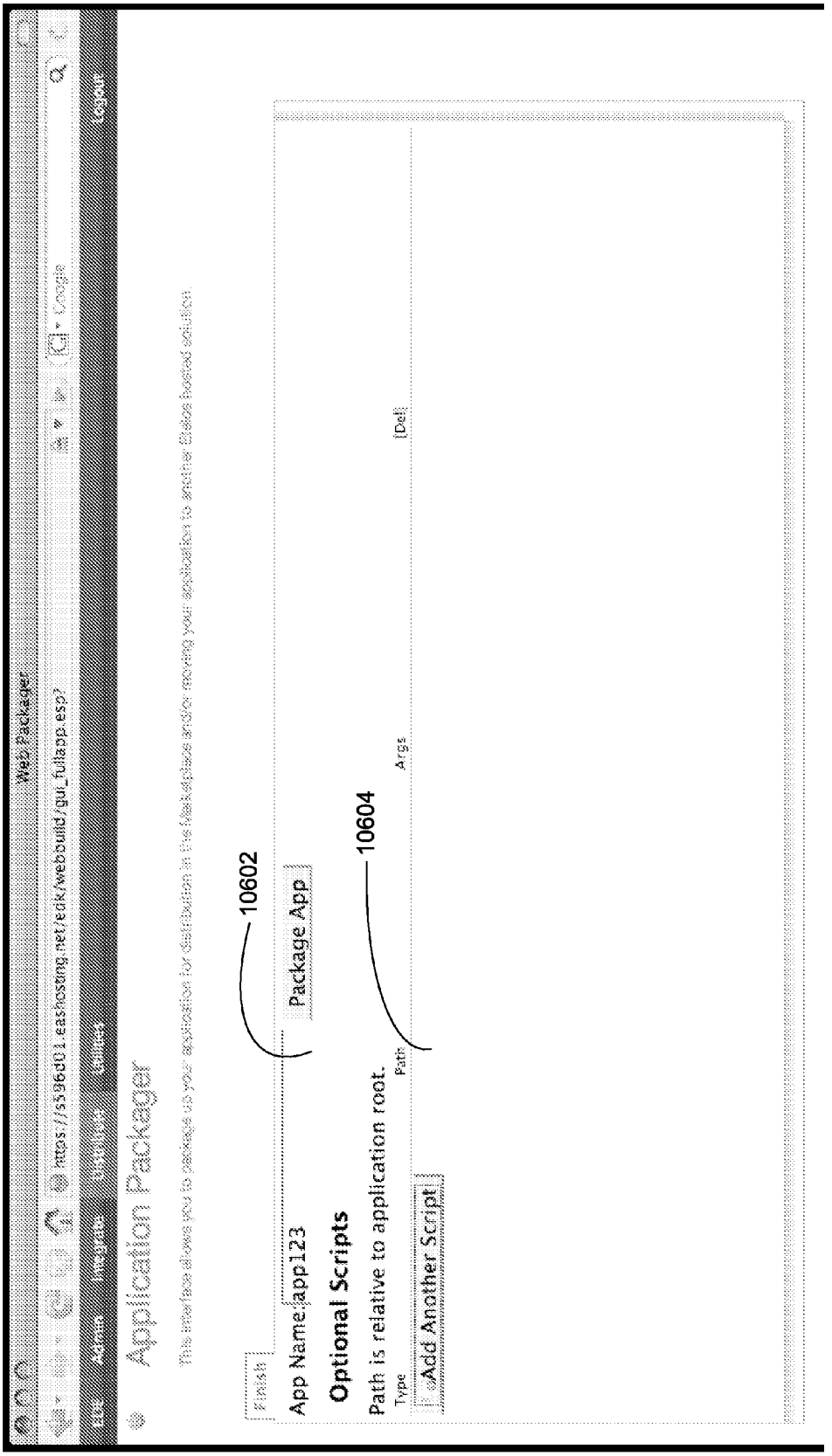

Web Packager
https://s596d01.eashosting.net/edk/webbuild/gui_fullapp.esp?
Google
Logout
Admin
Integrate
Upgrade
Utilities
Application Packager
This interface allows you to package up your application for distribution in the Marketplace and/or moving your application to another Elaios hosted solution.
Finish
App Name: app123
Package App
Optional Scripts
Path is relative to application root.
Type
Path
Args
[Del]
Add Another Script
10602
10604
10600

Figure 106

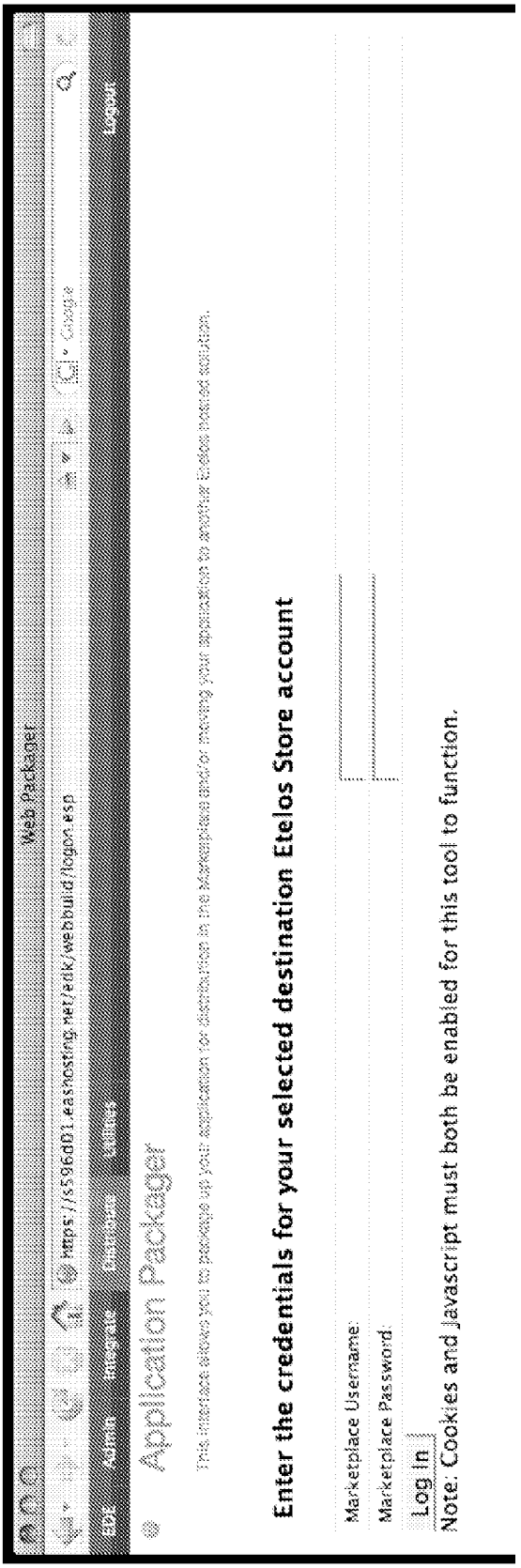

Web Packager
https://s596d01.easnosting.net/edk/webbuild/logon.esp
Google
Admin
Integrate
Publish
Edition
Logout
Application Packager
This interface allows you to package up your application for distribution in the Marketplace and/or moving your application to another Etelos hosted solution.
Enter the credentials for your selected destination Etelos Store account
Marketplace Username:
Marketplace Password:
Log In
Note: Cookies and Javascript must both be enabled for this tool to function.
10700

Figure 107

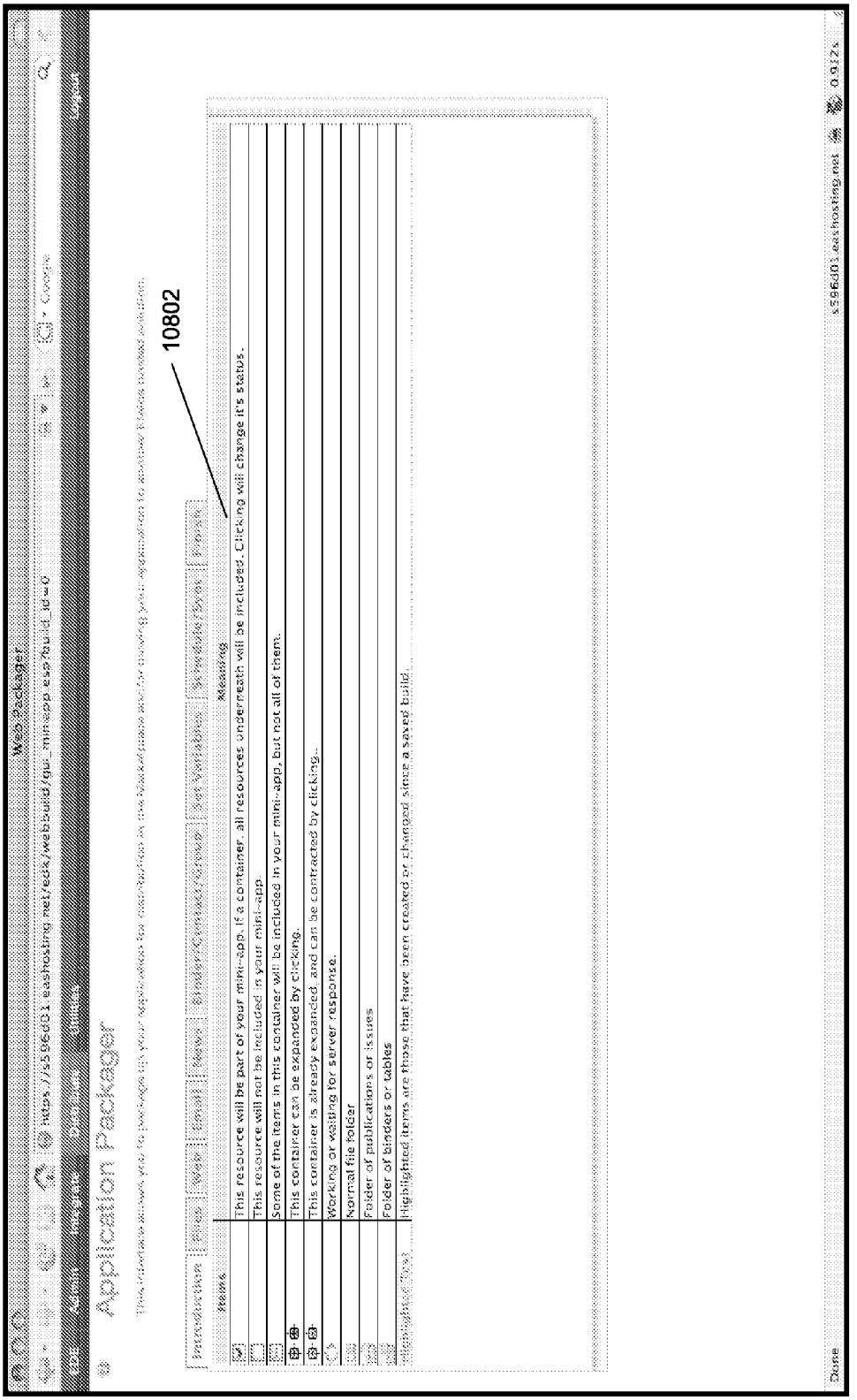

Web Packager
Application Packager
10802
Meaning
This resource will be part of your mini-app. If a container, all resources underneath will be included. Clicking will change it's status.
This resource will not be included in your mini-app
Some of the items in this container will be included in your mini-app, but not all of them.
This container can be expanded by clicking.
This container is already expanded, and can be contracted by clicking.
Working or waiting for server response.
Normal file folder
Folder of publications or issues
Folder of binders or tables
Highlighted items are those that have been created or changed since a saved build.
10800

Figure 108

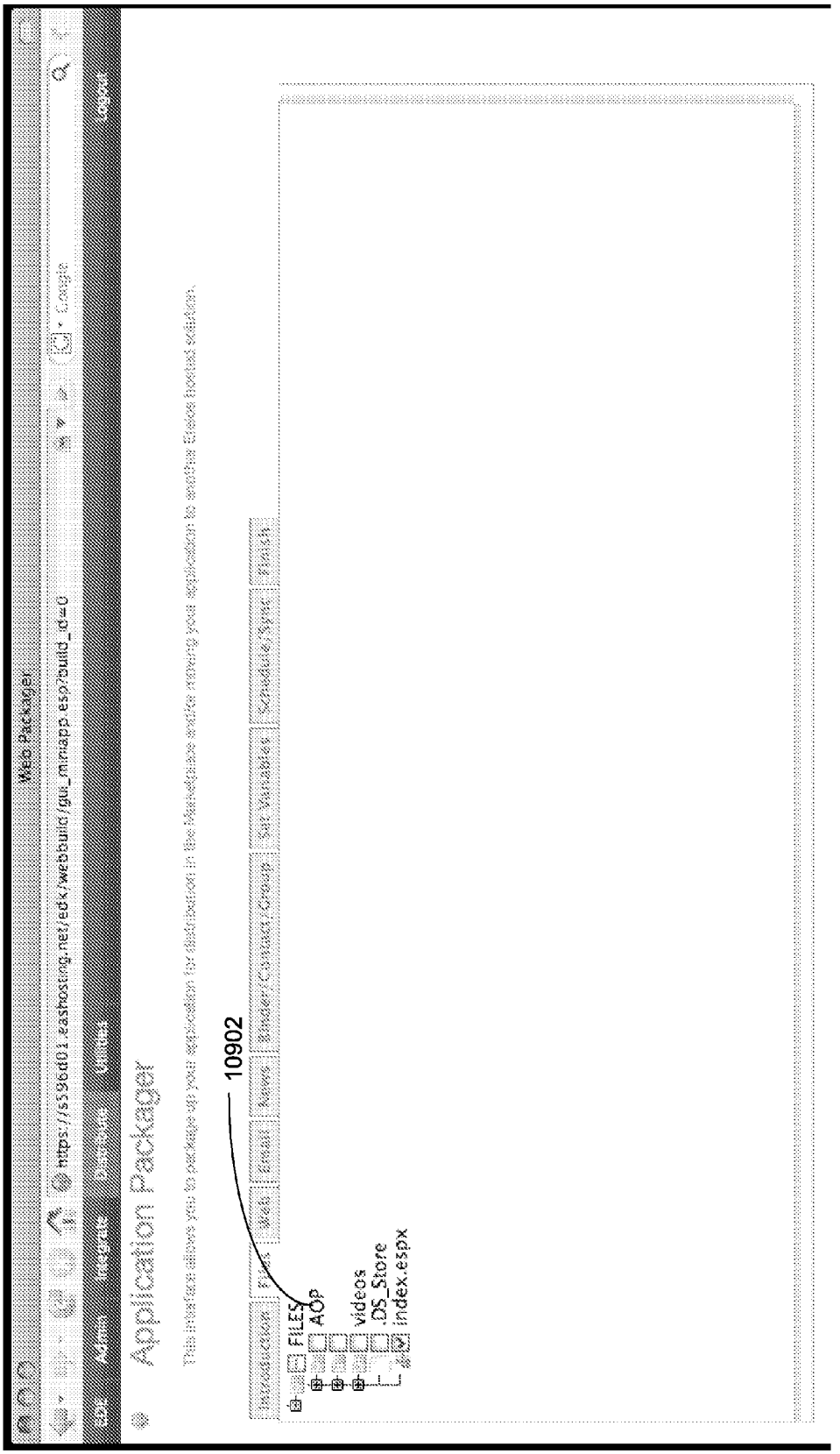

Web Packager
Application Packager
10902
FILES
AOP
videos
.DS_Store
index.espx
10900

Figure 109

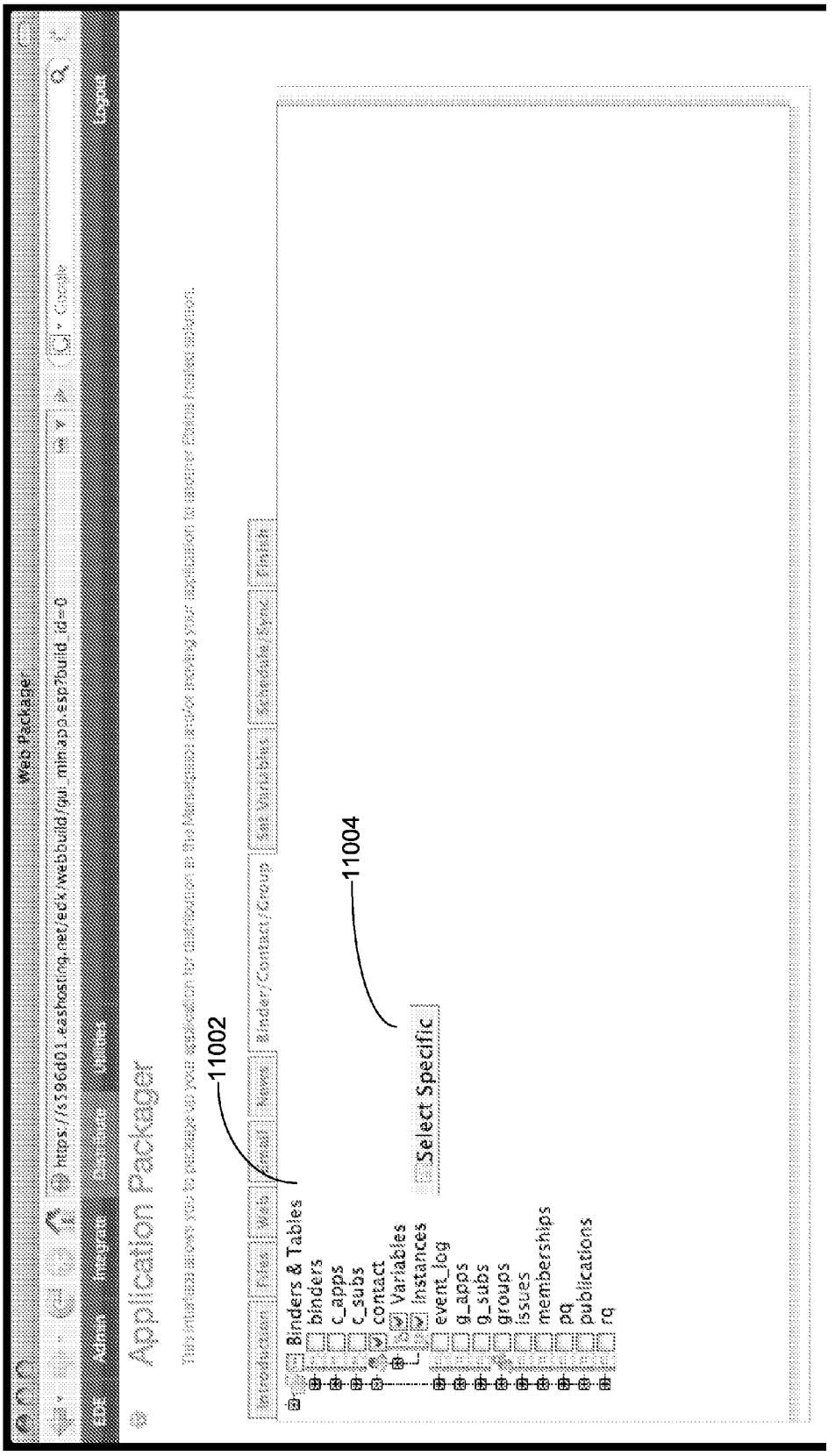
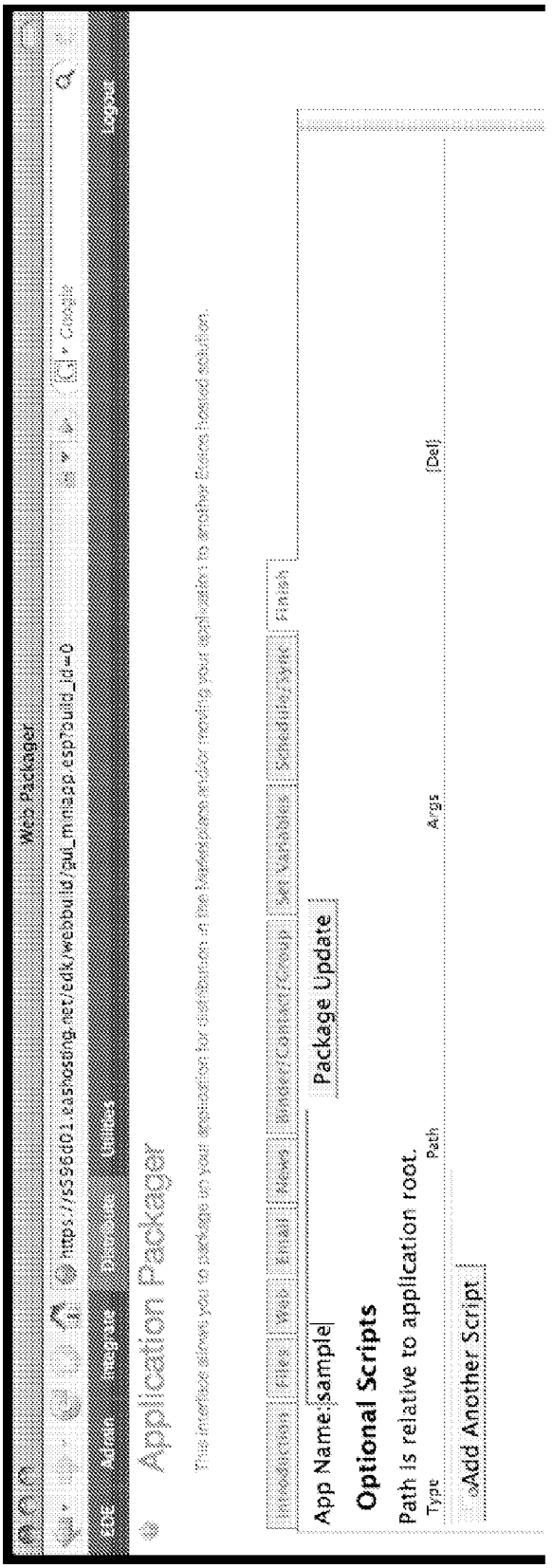

Web Packager
Application Packager
Introduction
Files
Web
Email
News
Binder/Contact/Group
Save Variables
Schedule Sync
Finish
Binders & Tables
binders
c_apps
c_subs
contact
Variables
Instances
event_log
g_apps
g_subs
groups
issues
memberships
pq
publications
rq
Select Specific
11002
11004
11000

Figure 110



11100

Figure 111

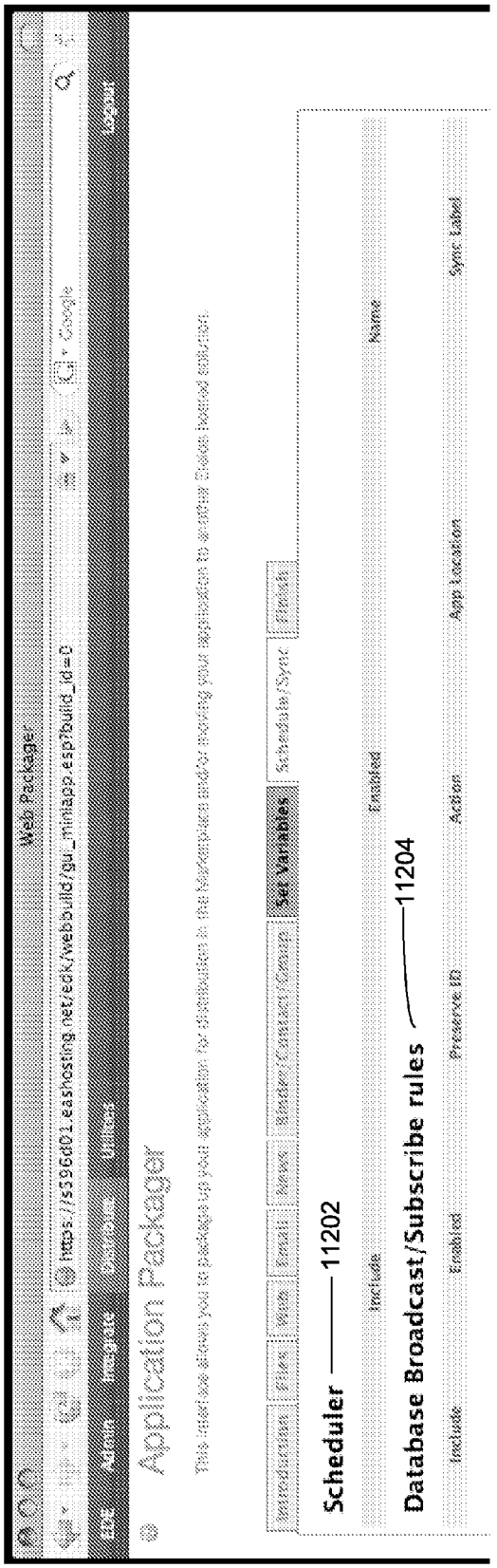

Web Packager
Application Packager
Scheduler
11202
Database Broadcast/Subscribe rules
11204
11200

Figure 112

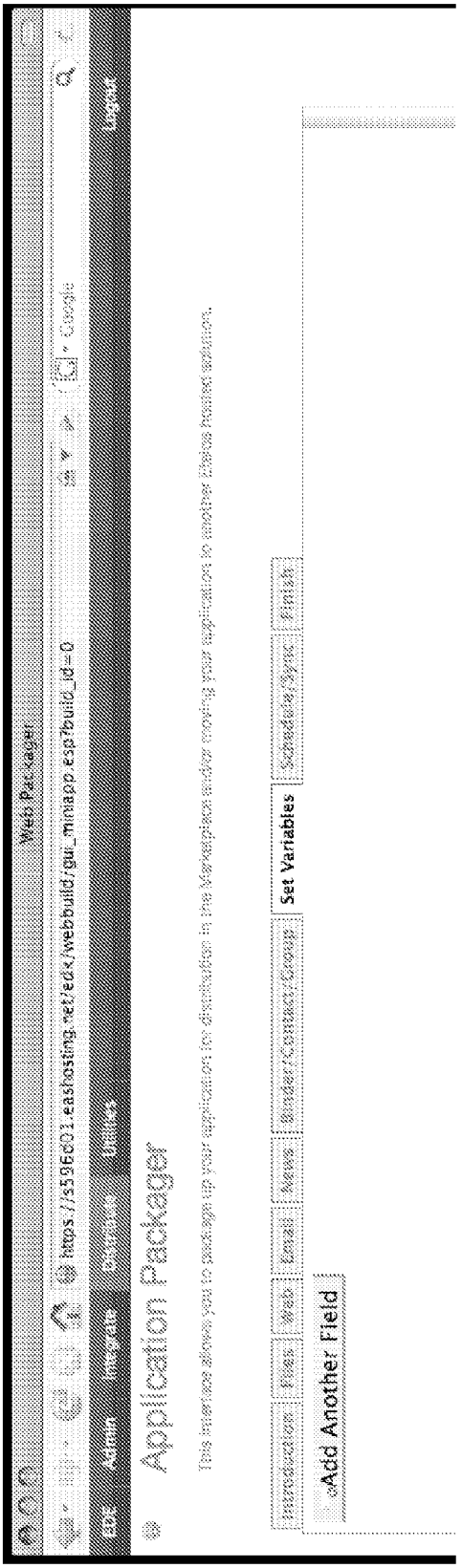

Web Packager
https://s3866c01.easthosting.net/edx/webbuild/gui_miniapp.esp?build_id=0
Admin
Integrate
Applications
Utilities
Logout
Application Packager
This interface allows you to package up your application for distribution in the Marketplace and/or making your application to another Edge Master system.
Introduction
Files
Web
Email
News
Binder/Contact/Group
Set Variables
Schedule Sync
Finish
Add Another Field
11300

Figure 113

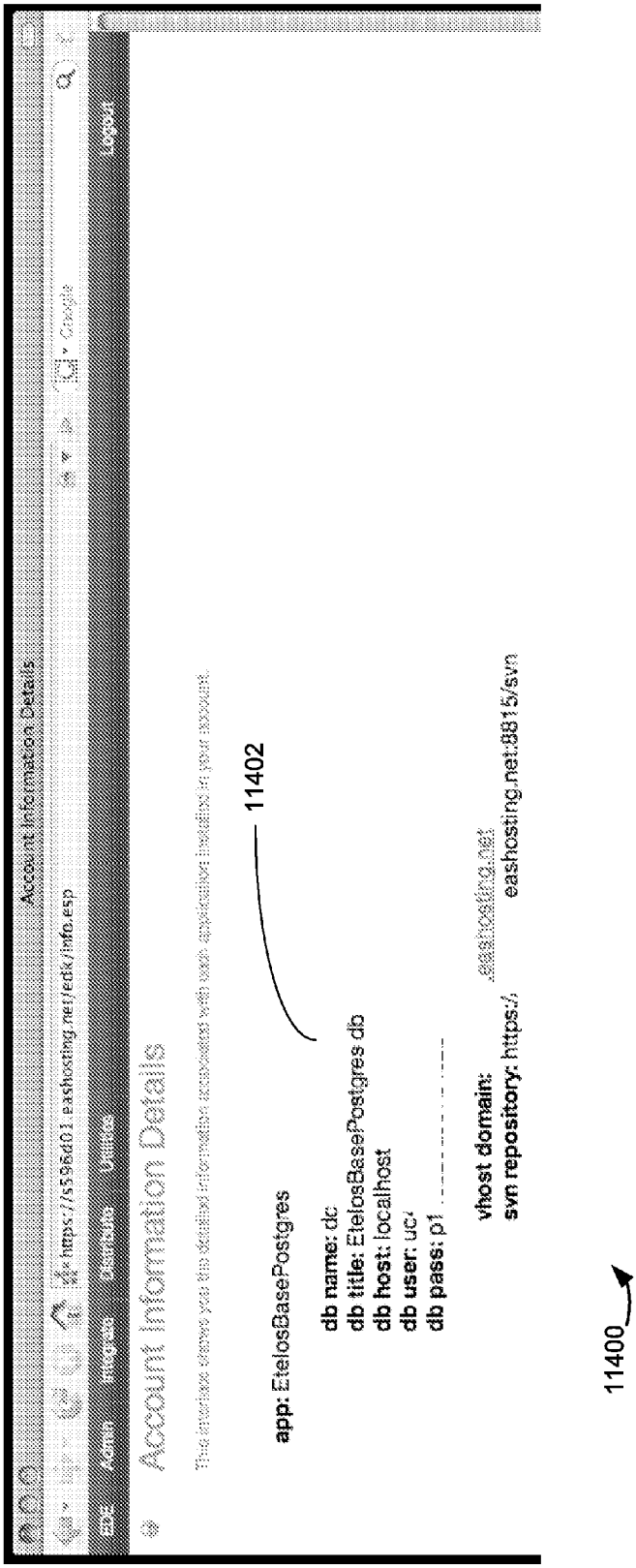

Account Information Details
https://s596d01.eashosting.net/edk/info.esp
Google
EDE
Admin
Integrate
Distribute
Utilities
Logout
Account Information Details
app: EtelosBasePostgres
db name: dc
db title: EtelosBasePostgres db
db host: localhost
db user: uc/
db pass: p1
vhost domain: .eashosting.net
svn repository: https:/ eashosting.net:8815/svn
11402
11400

Figure 114

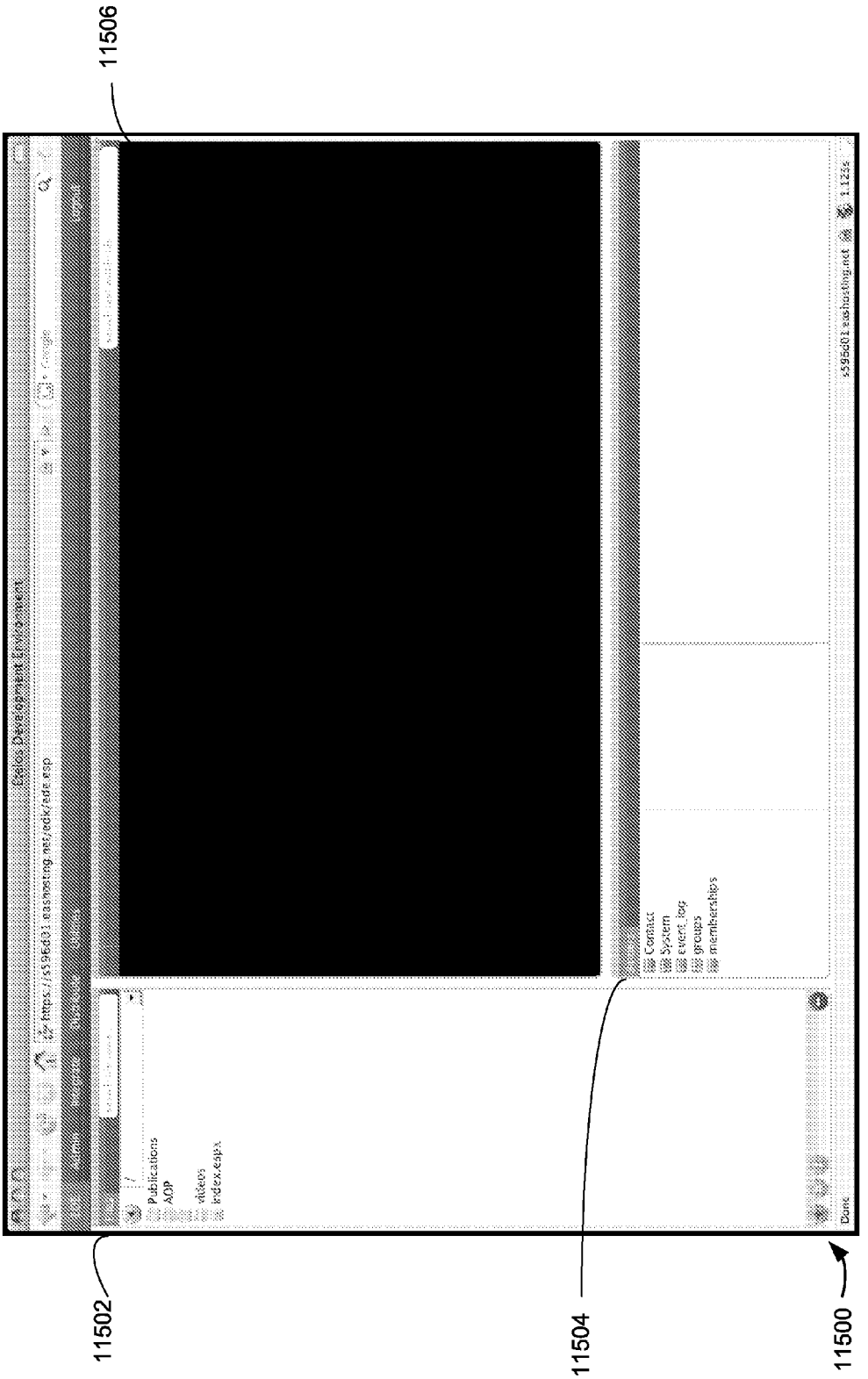

11506
11502
11504
11500
Publications
ADP
videos
index.espx
Contact
System
event_log
groups
memberships

Figure 115

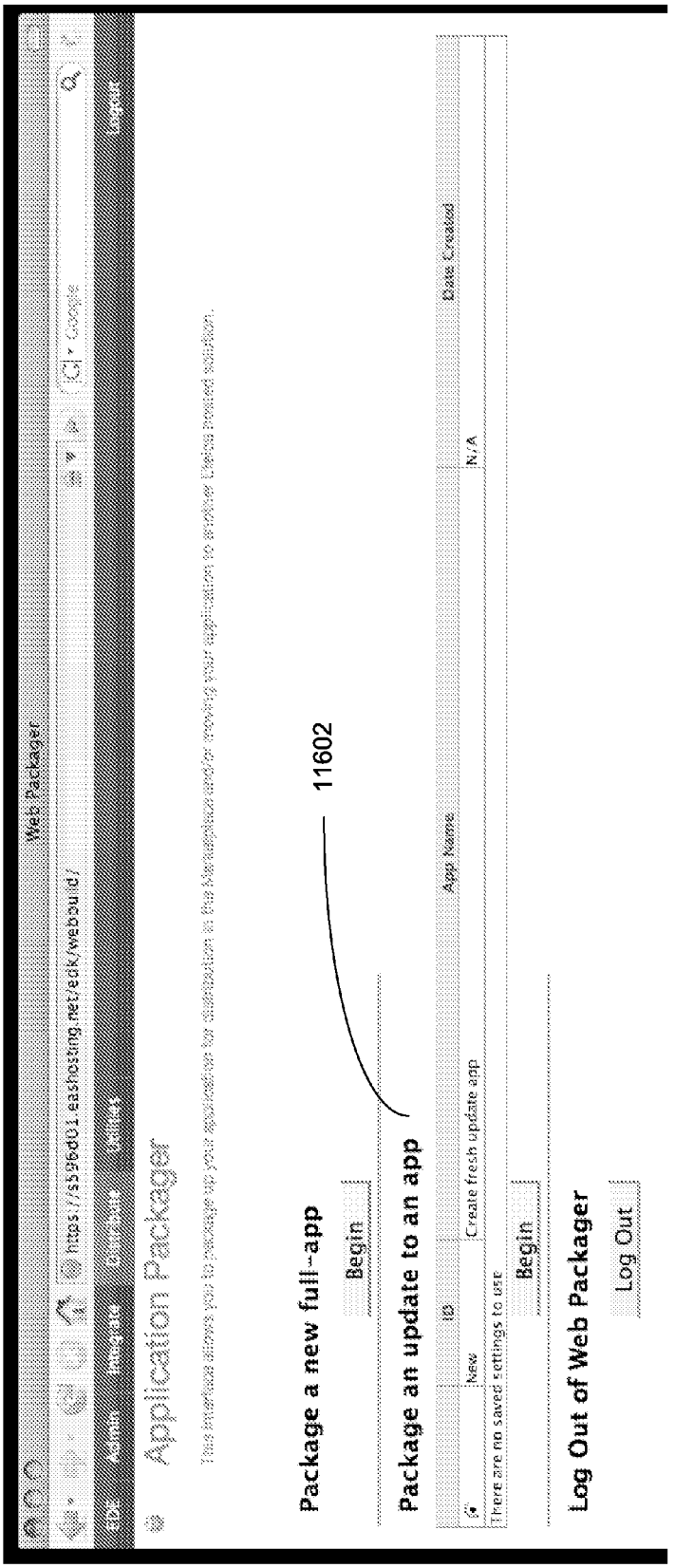

Web Packager
Application Packager
Package a new full-app
Begin
Package an update to an app
ID
App Name
Date Created
New
Create fresh update app
N/A
There are no saved settings to use
Begin
Log Out of Web Packager
Log Out
11602
11600

Figure 116

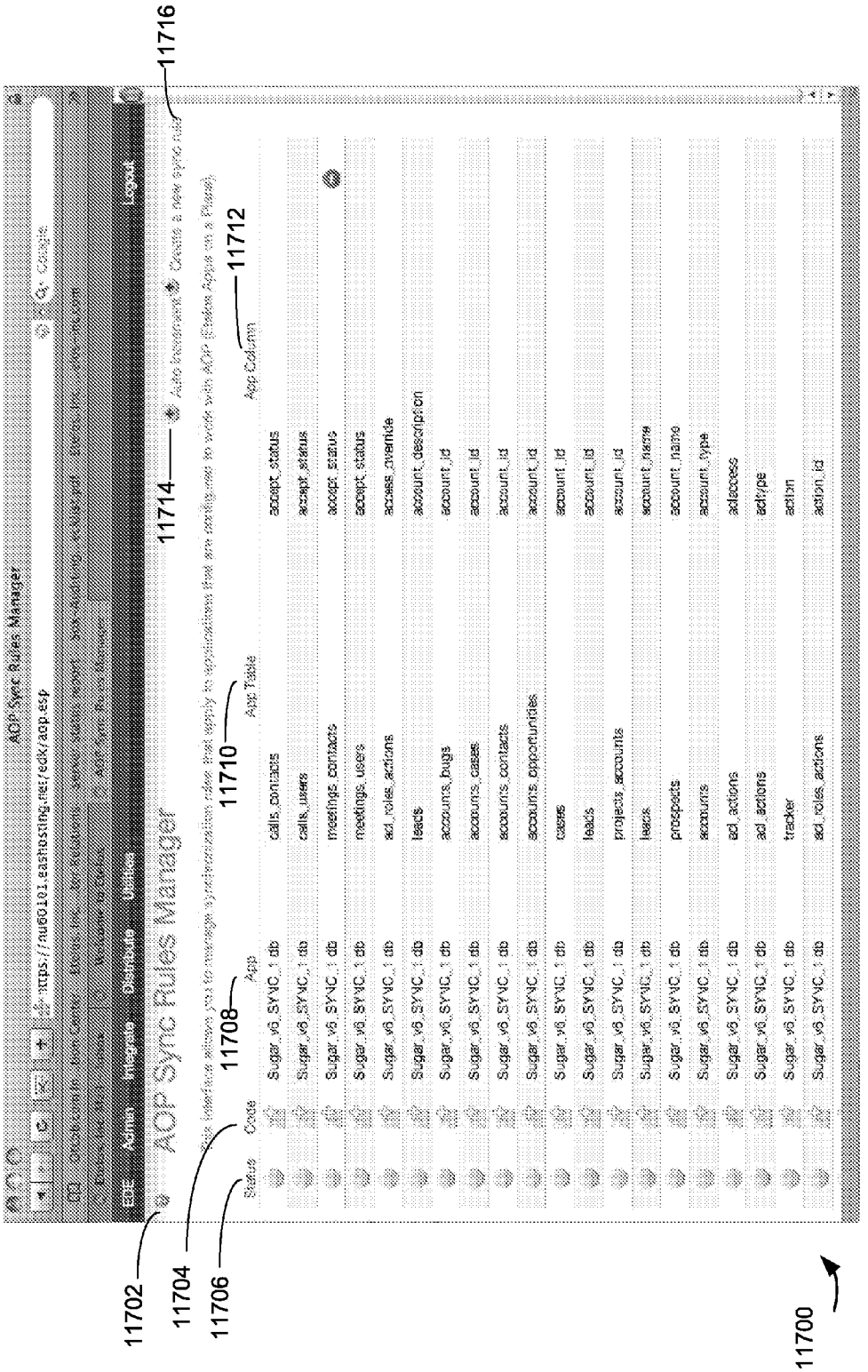

AOP Sync Rules Manager
AOP Sync Rules Manager
11702
11704
11706
11708
11710
11712
11714
11716
11700

Figure 117

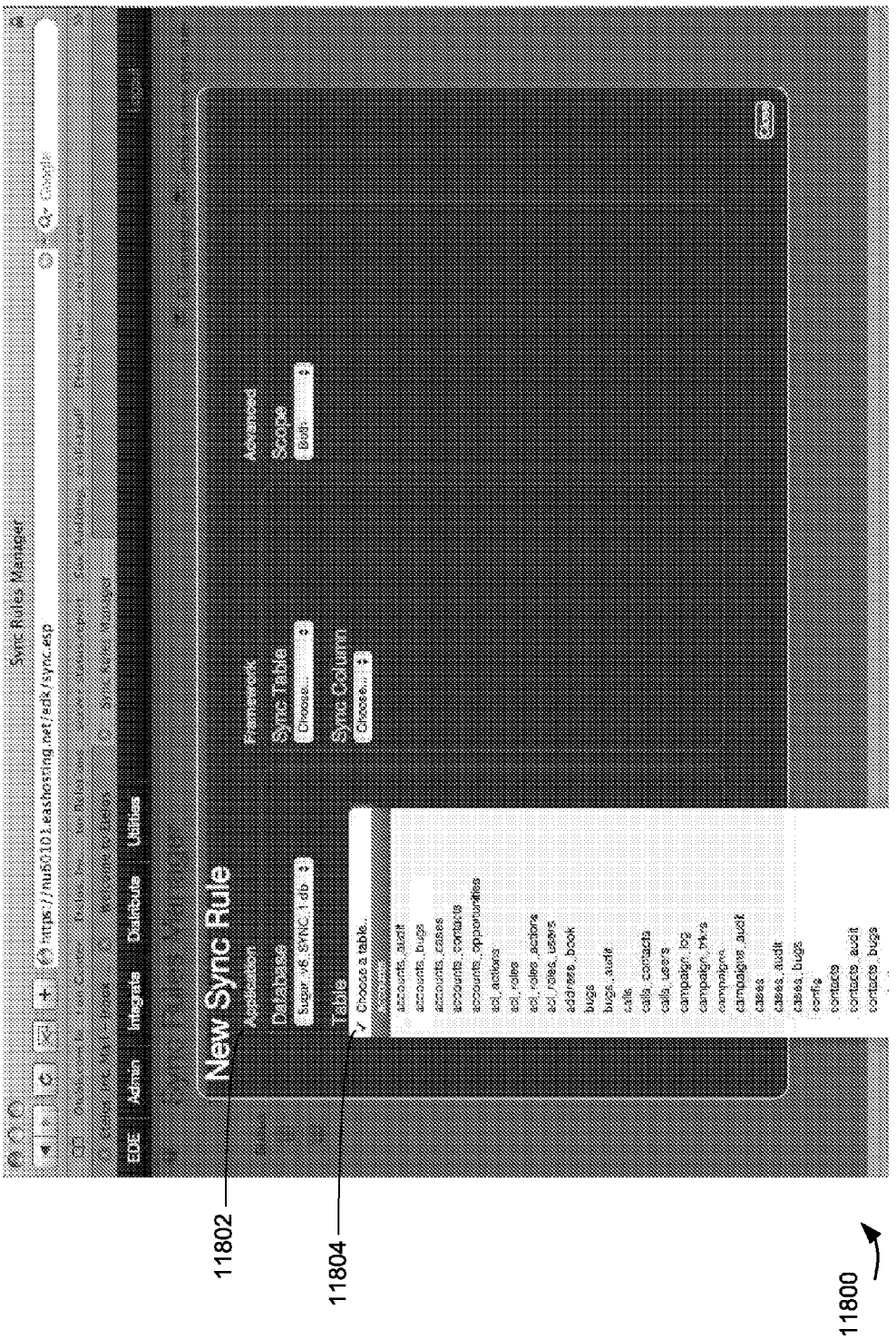

Sync Rules Manager
https://nu60101.eashosting.net/eak/sync.esp
EDE
Admin
Integrate
Distribute
Utilities
New Sync Rule
Application
Database
sugar_v6_SYNC_1.db
Table
Framework
Sync Table
Choose...
Sync Column
Choose...
Advanced
Scope
Both
Choose a table...
accounts_audit
accounts_bugs
accounts_cases
accounts_contacts
accounts_opportunities
acl_actions
acl_roles
acl_roles_actions
acl_roles_users
address_book
bugs
bugs_audit
calls
calls_contacts
calls_users
campaign_log
campaign_trkrs
campaigns
campaigns_audit
cases
cases_audit
cases_bugs
config
contacts
contacts_audit
contacts_bugs
11802
11804
11800

Figure 118

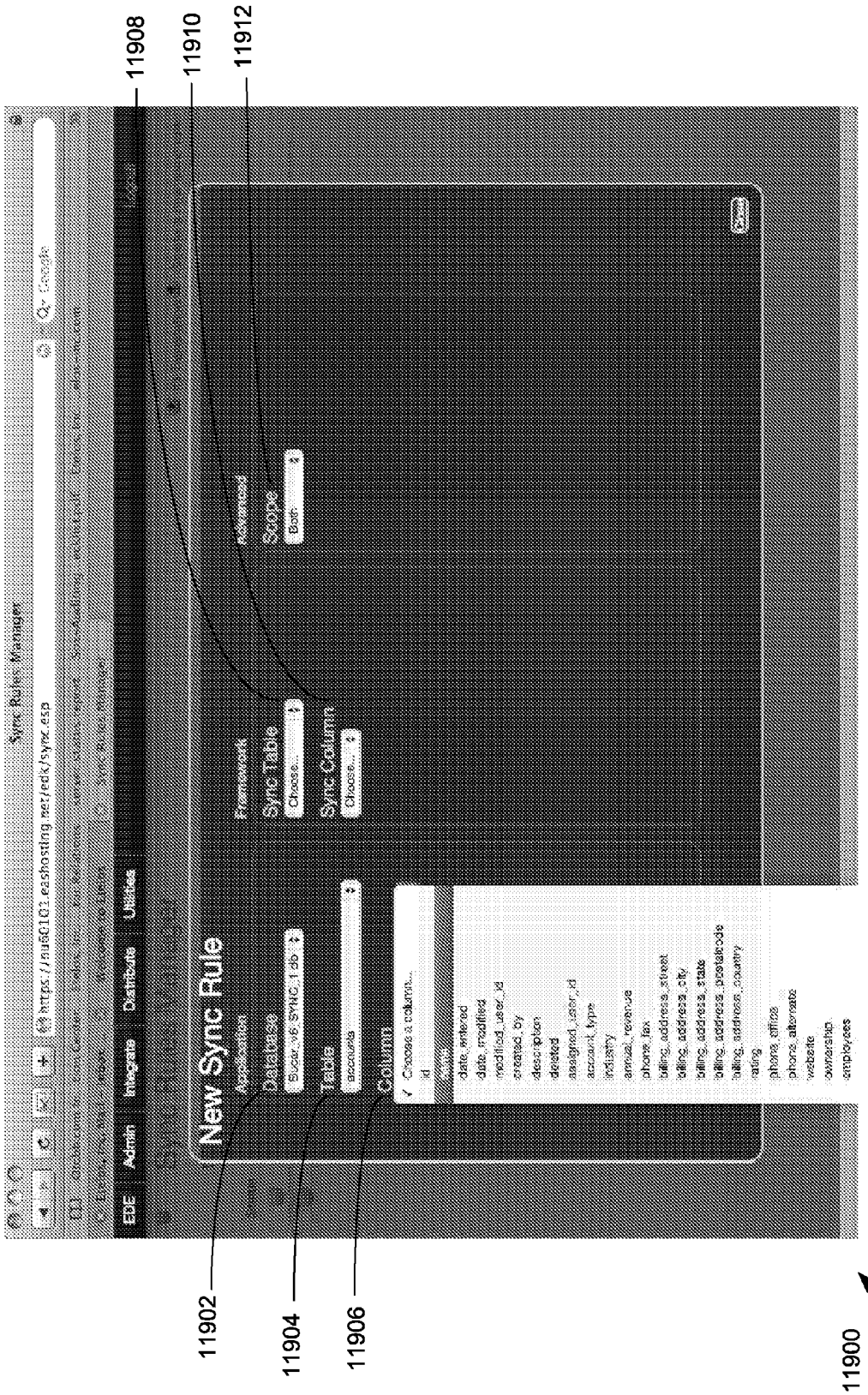

11908
11910
11912
11902
11904
11906
11900
Sync Rules Manager
EBE
Admin
Integrate
Distribute
Utilities
New Sync Rule
Application
Framework
Advanced
Database
Sync Table
Scope
Table
Sync Column
Column
Choose a column...
id
name
date_entered
date_modified
modified_user_id
created_by
description
deleted
assigned_user_id
account_type
industry
annual_revenue
phone_fax
billing_address_street
billing_address_city
billing_address_state
billing_address_postalcode
billing_address_country
rating
phone_office
phone_alternate
website
ownership
employees

Figure 119

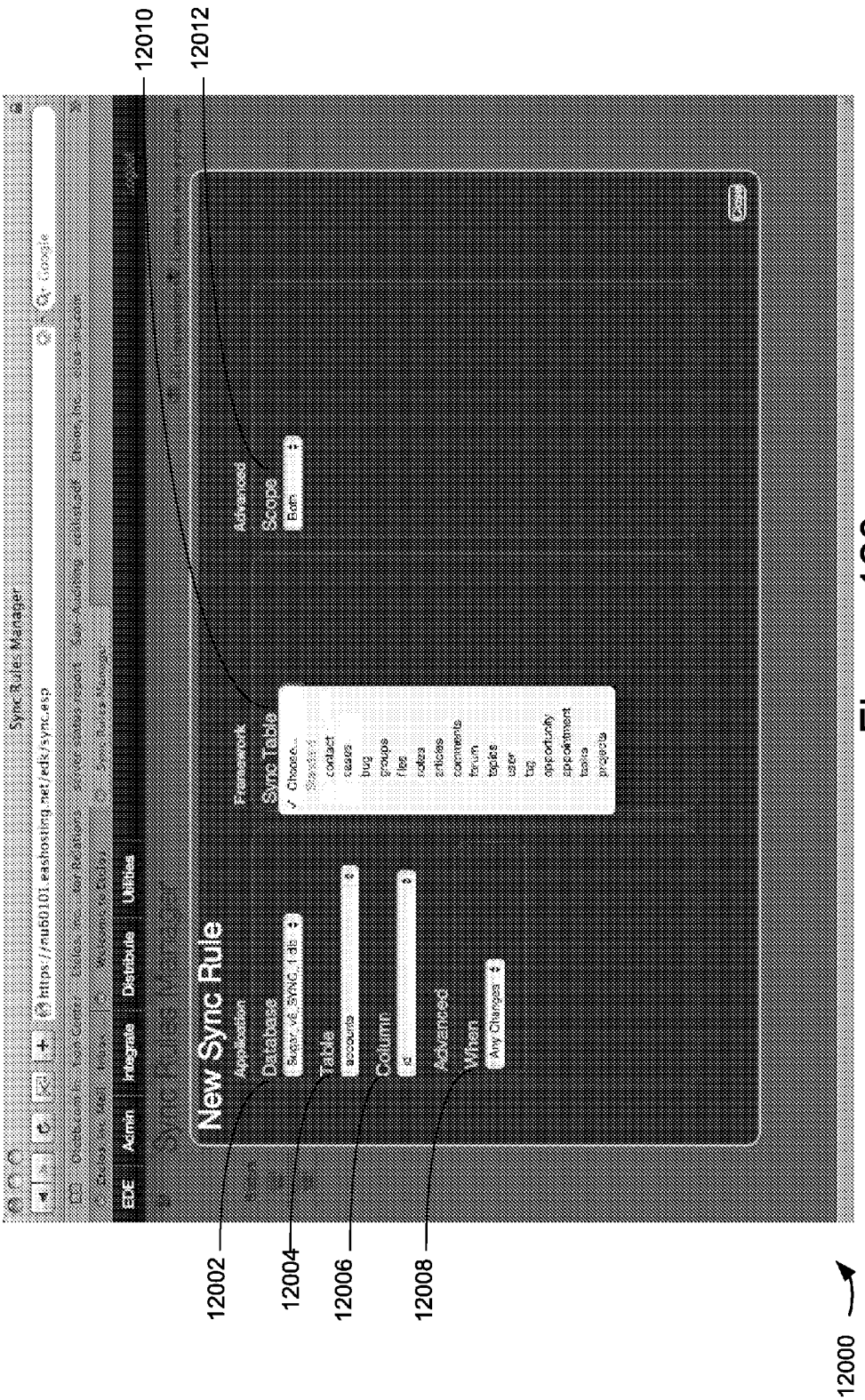

12010
12012
Sync Rules Manager
EDE
Admin
Integrate
Distribute
Utilities
New Sync Rule
Application
Database
Table
Column
Advanced
When
Any Changes
accounts
id
Framework
Sync Table
Advanced
Scope
Both
Choose...
contact
cases
bug
groups
files
notes
articles
comments
forum
topics
user
tag
opportunity
appointment
tasks
projects
12002
12004
12006
12008
12000

Figure 120

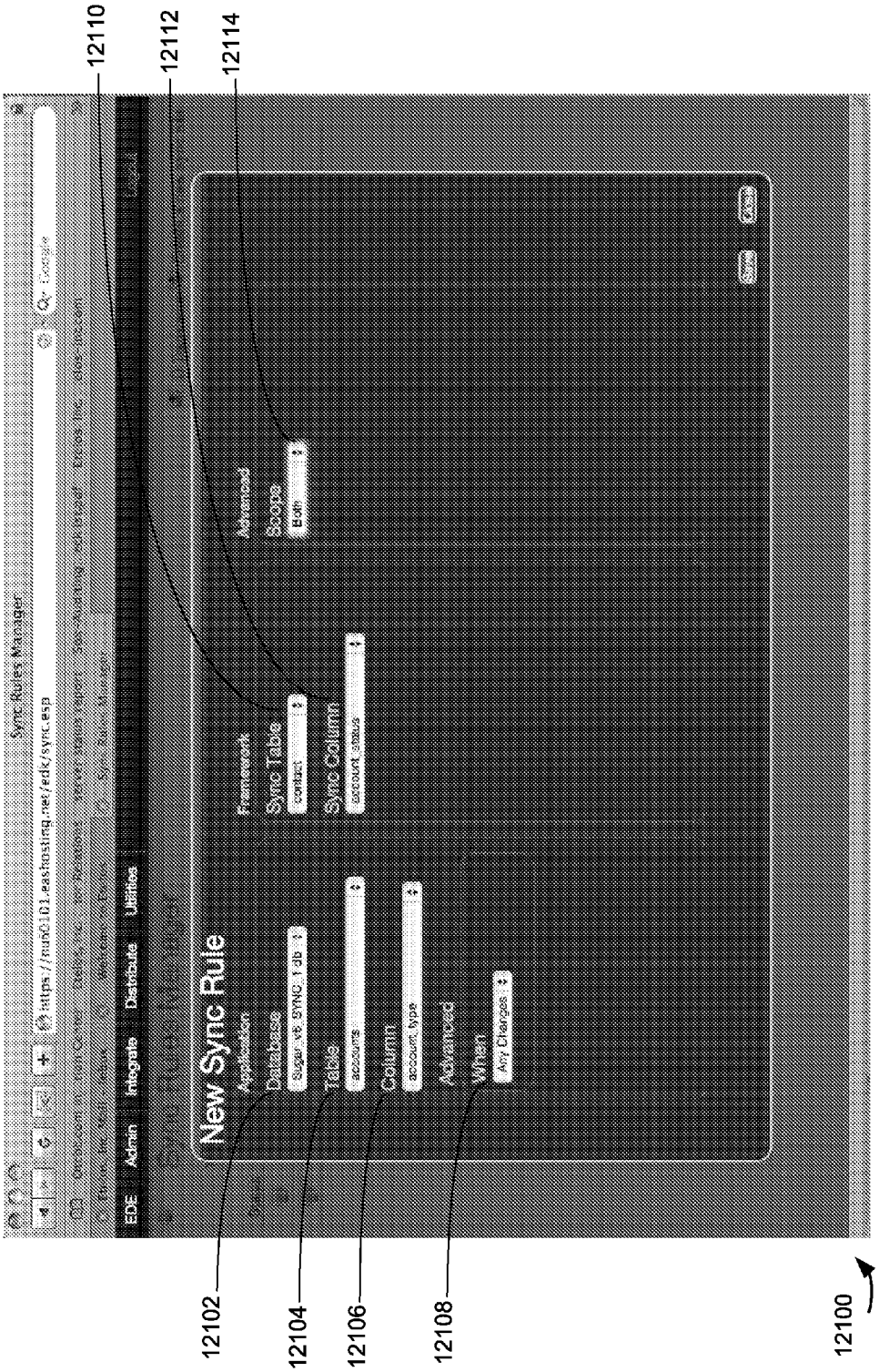

12110
12112
12114
Sync Rules Manager
EDE
Admin
Integrate
Distribute
Utilities
New Sync Rule
Application
Database
Sugar v6 SYNC_1 db
Table
accounts
Column
account_type
Advanced
When
Any Changes
Framework
Sync Table
contact
Sync Column
account_status
Advanced
Scope
Both
12102
12104
12106
12108
12100

Figure 121

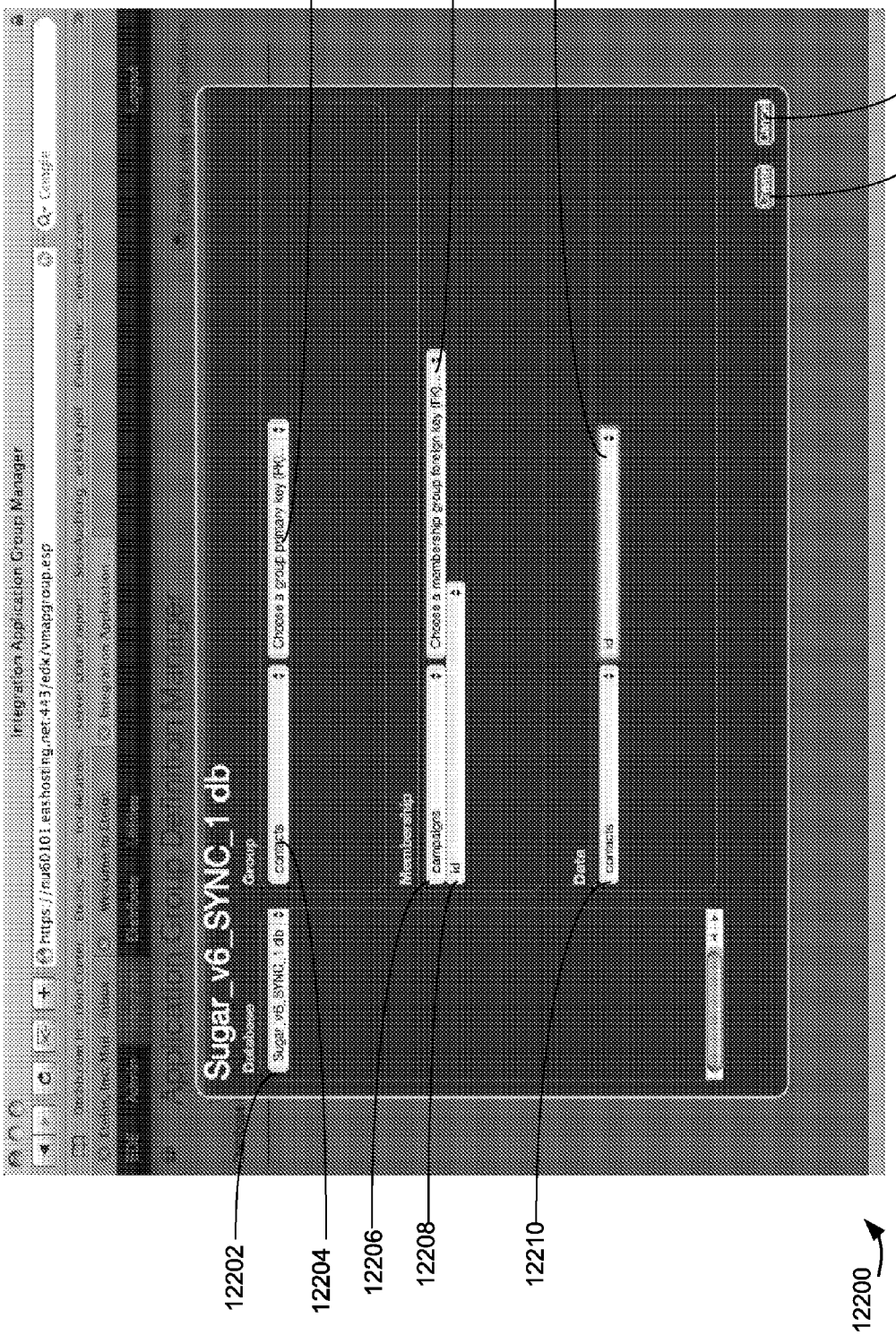

Integration Application Group Manager
Sugar_v6_SYNC_1 db
Database
Group
Membership
Data
12202
12204
12206
12208
12210
12212
12214
12216
12218
12220
12200

Figure 122

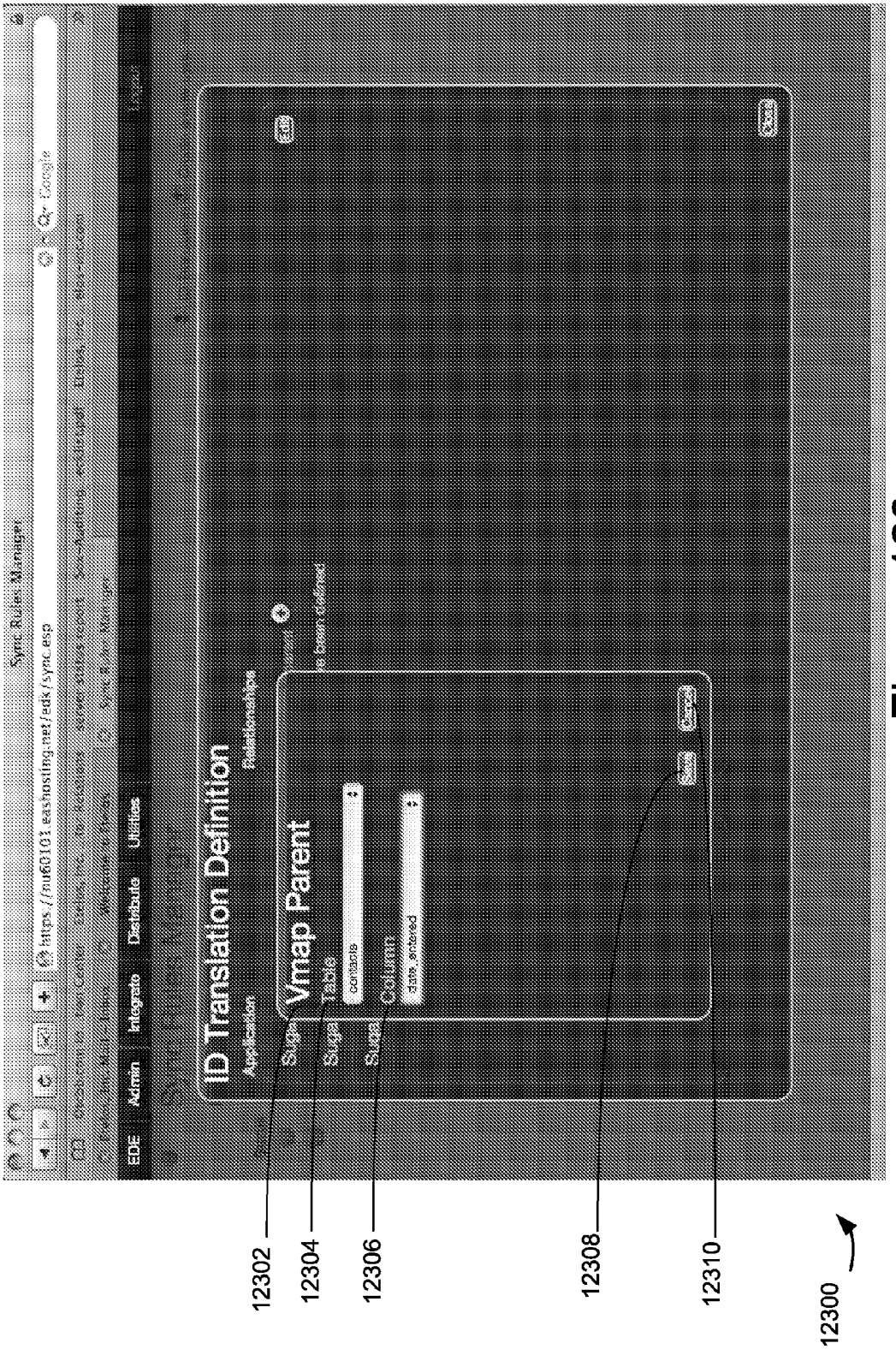

Sync Rules Manager
EDE
Admin
Integrate
Distribute
Utilities
ID Translation Definition
Application
Relationships
Vmap Parent
Table
contacts
Column
date_entered
Save
Cancel
12302
12304
12306
12308
12310
12300

Figure 123

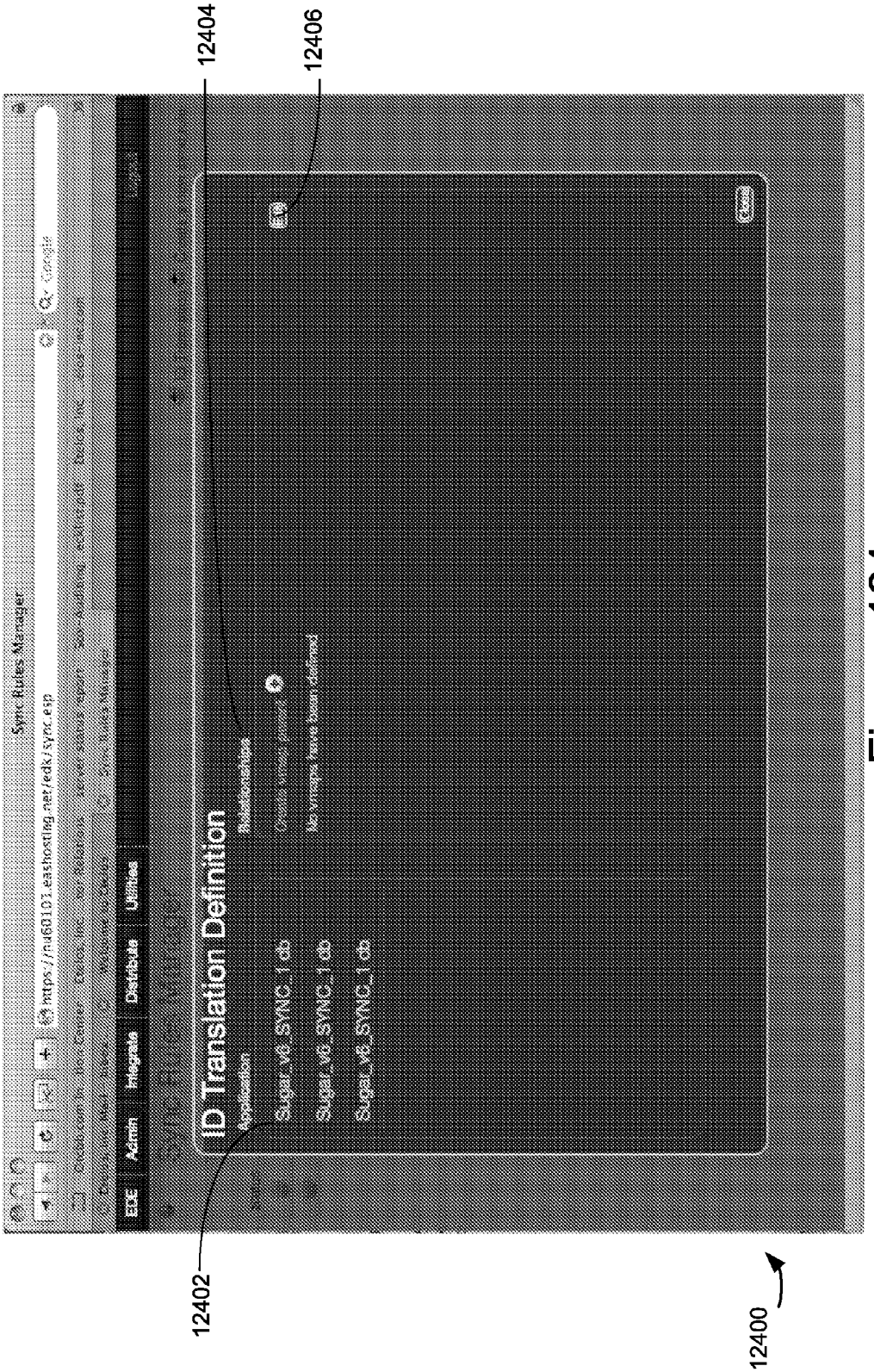

12404
12406
12402
12400
Sync Rules Manager
EDE
Admin
Integrate
Distribute
Utilities
ID Translation Definition
Application
Relationships
Sugar_v8_SYNC_1.db
Sugar_v8_SYNC_1.db
Sugar_v8_SYNC_1.db

Figure 124

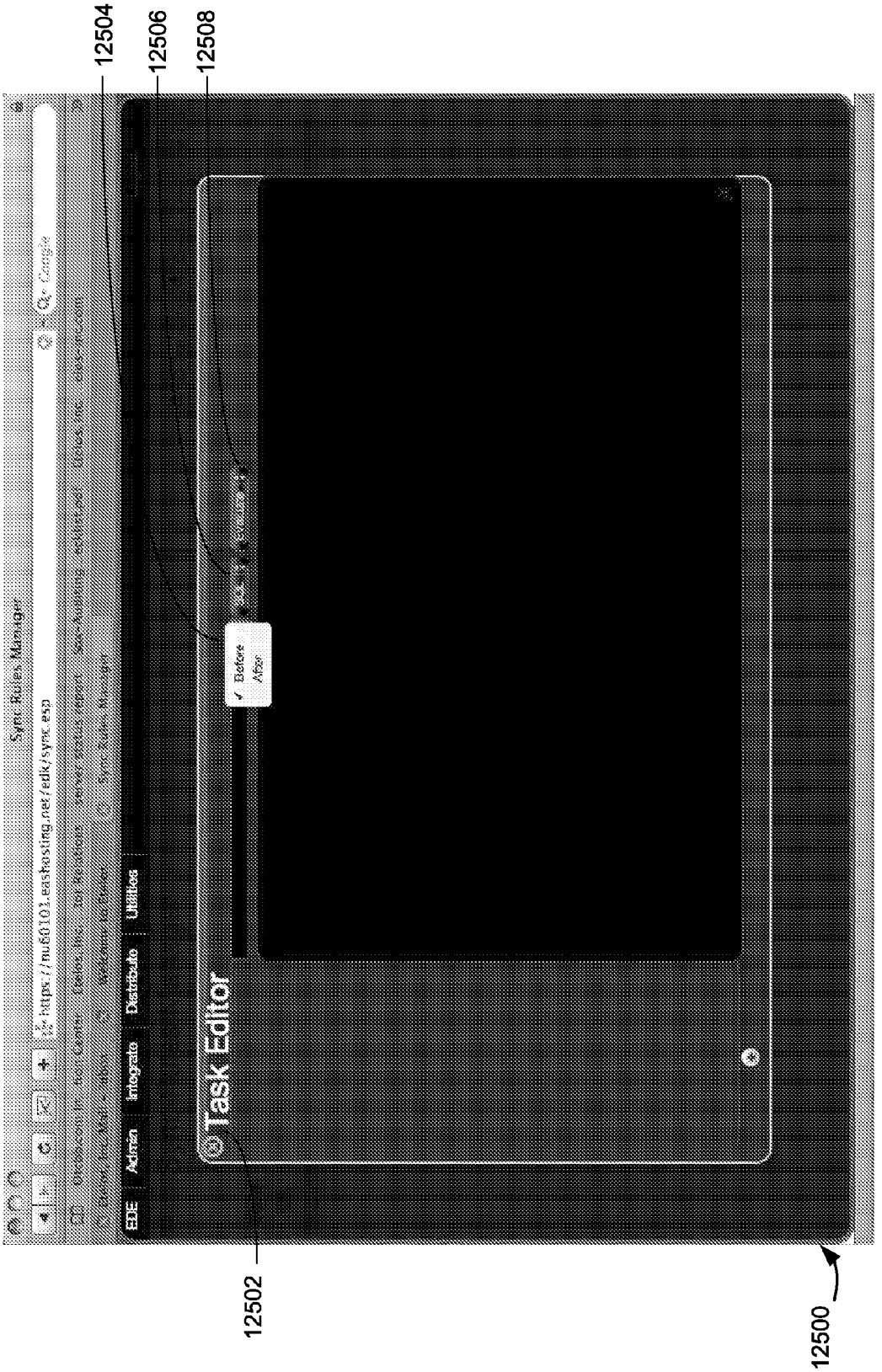

12504
12506
12508
12502
12500
Sync Rules Manager
EDE
Admin
Integrate
Distribute
Utilities
Task Editor
Before
After

Figure 125

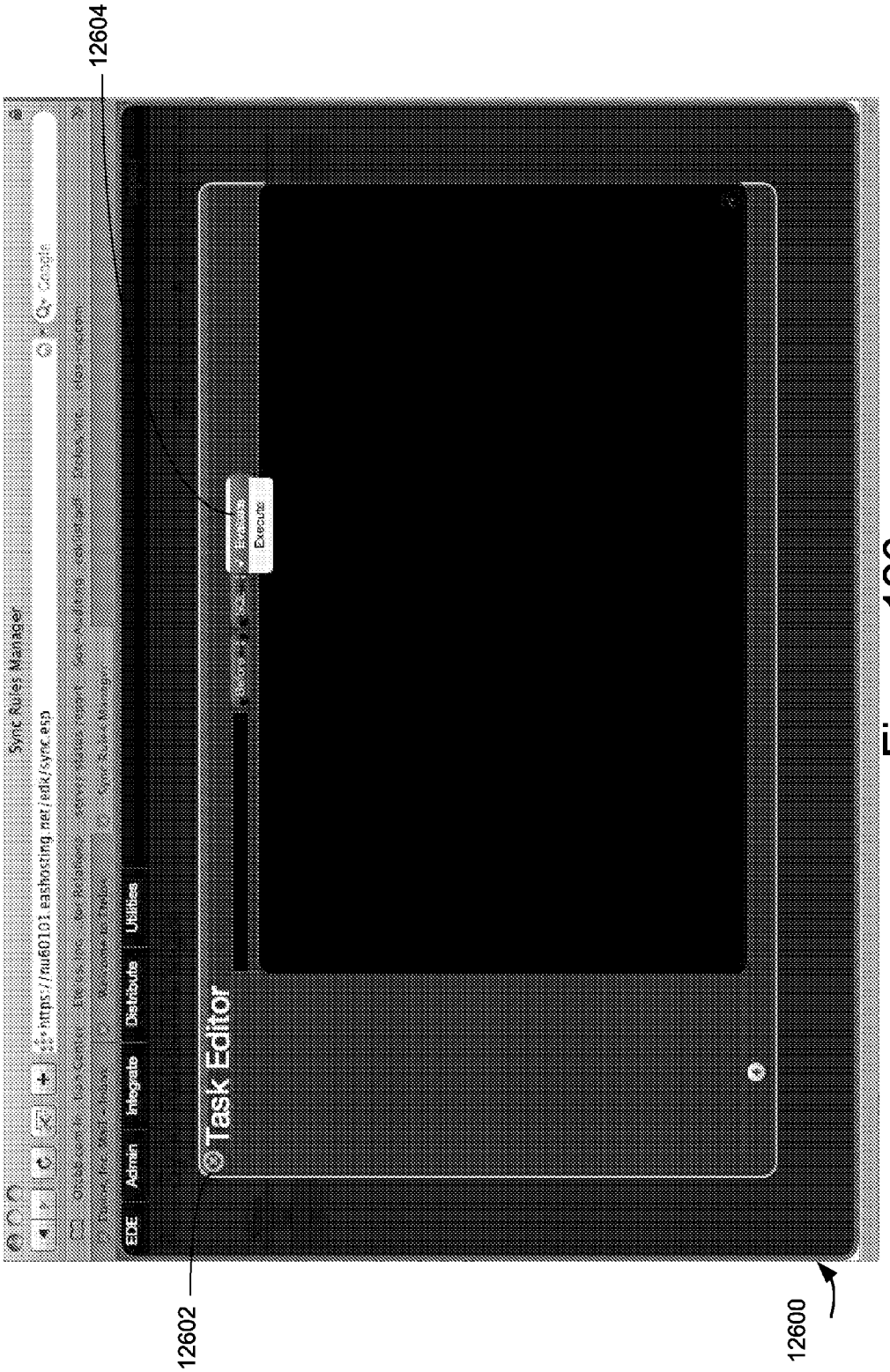

12604
Sync Rules Manager
https://ru60101.eashosting.net/edk/sync.esp
EDE
Admin
Integrate
Dashboards
Utilities
Task Editor
Execute
12602
12600

Figure 126

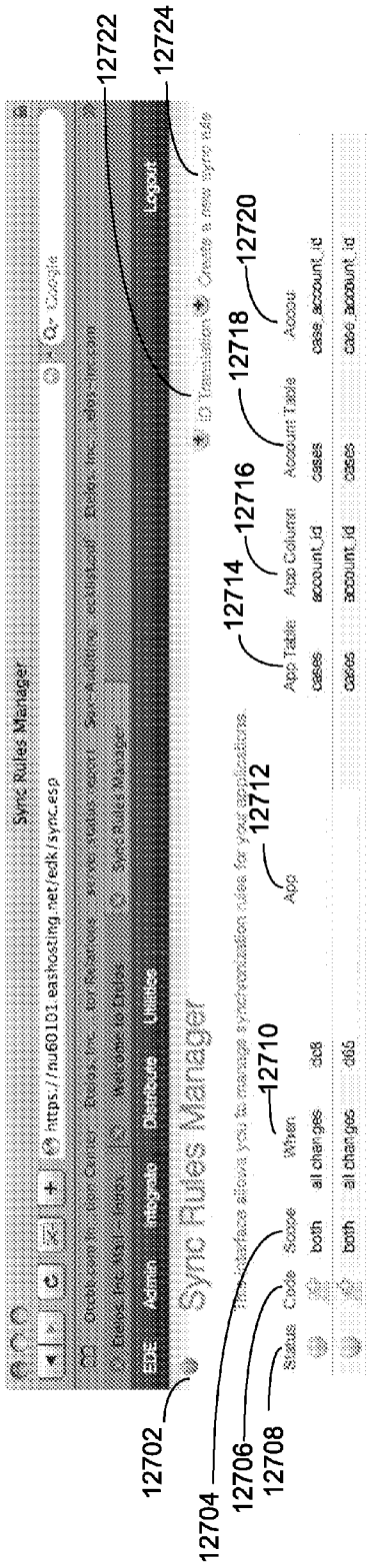
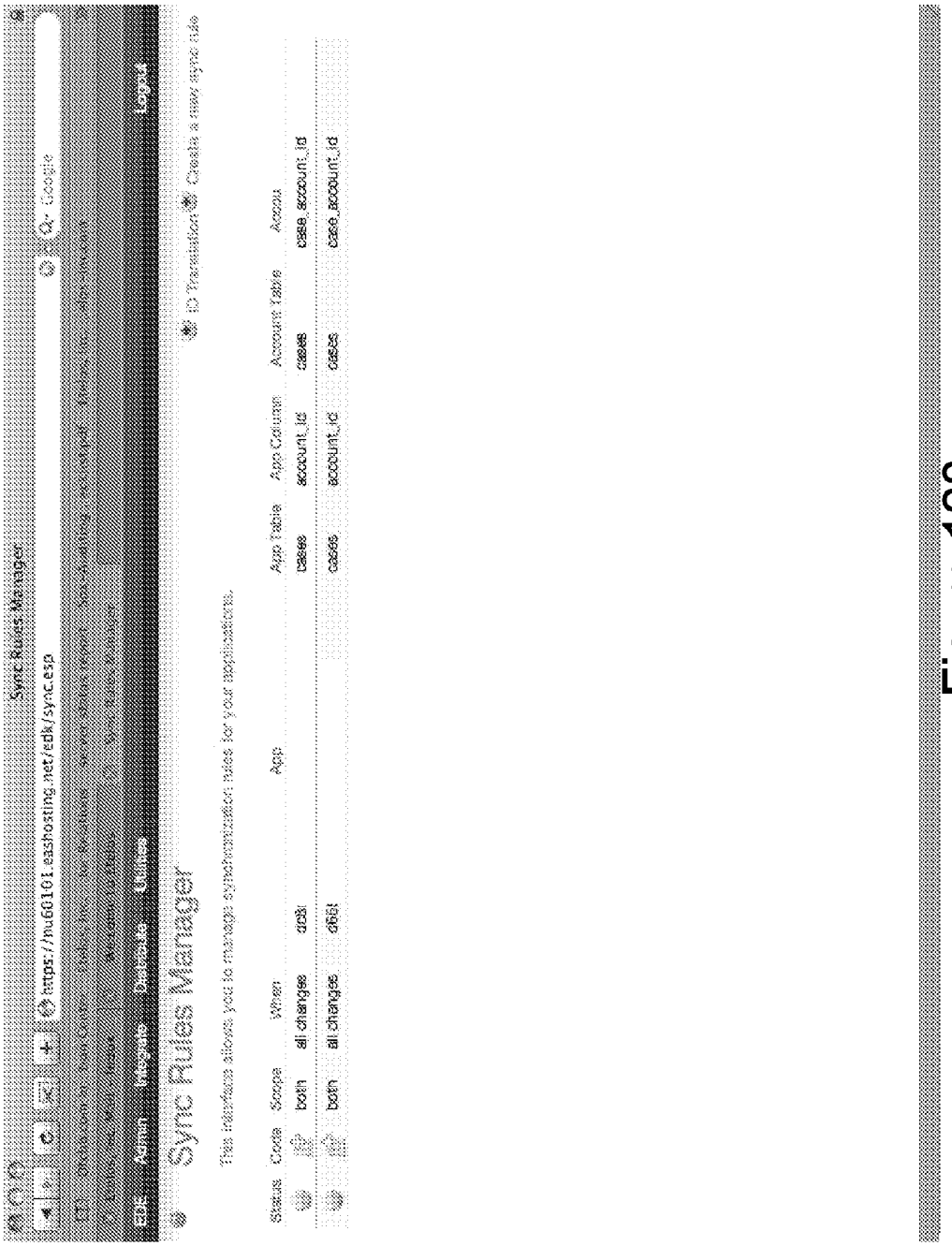


Figure 127

12700



12800

Figure 128

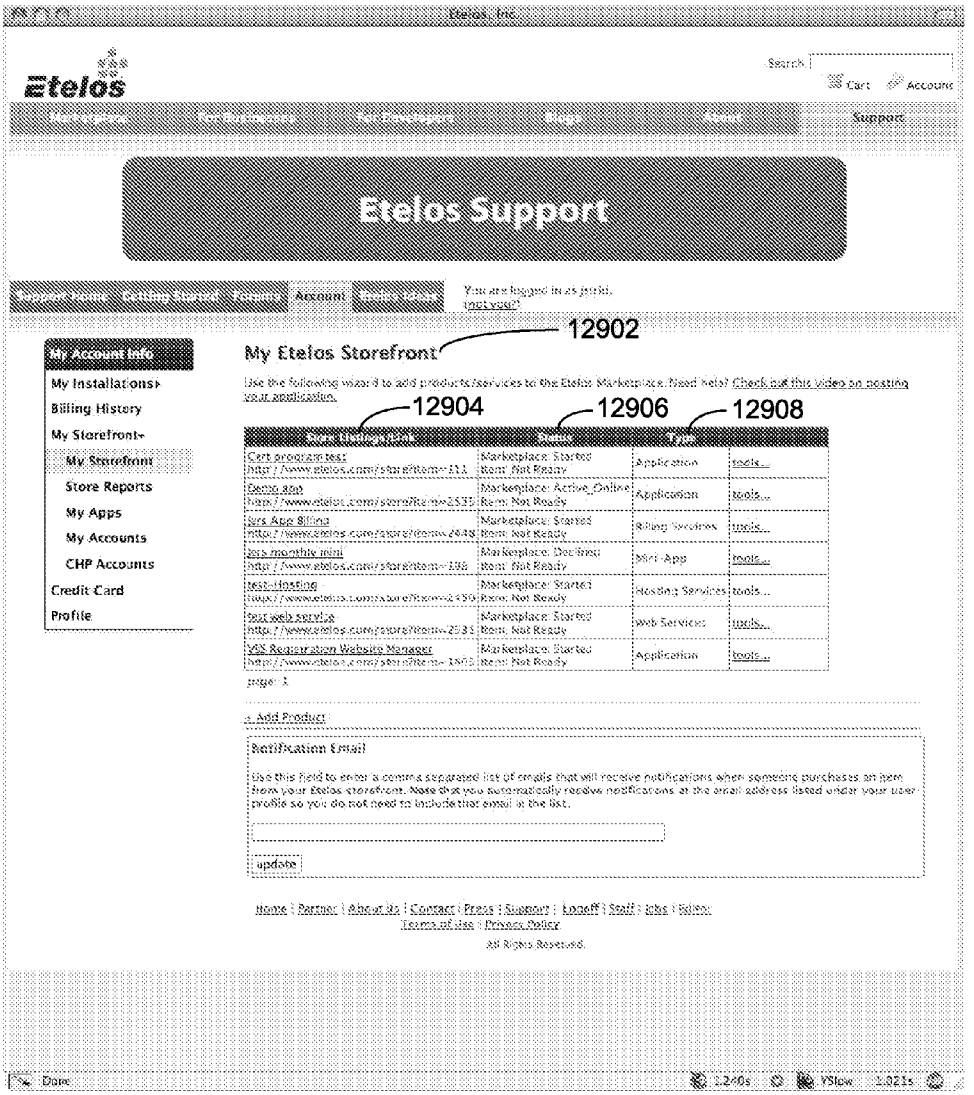
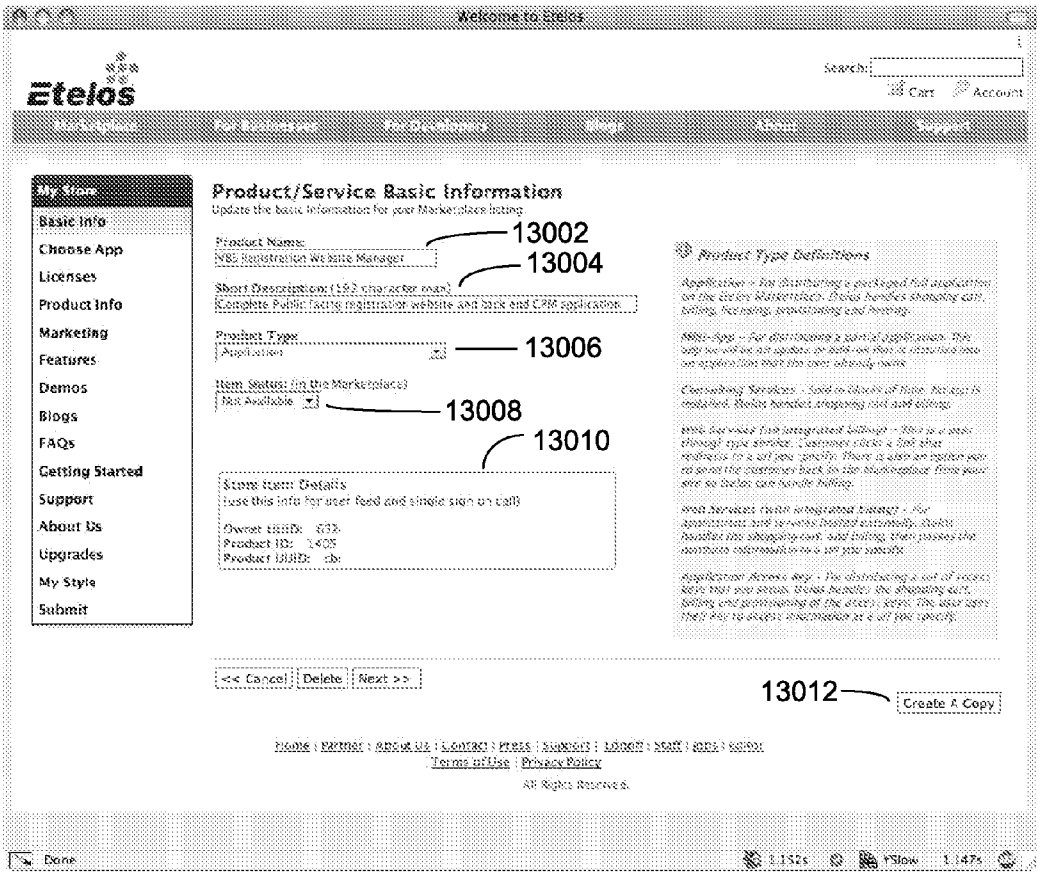
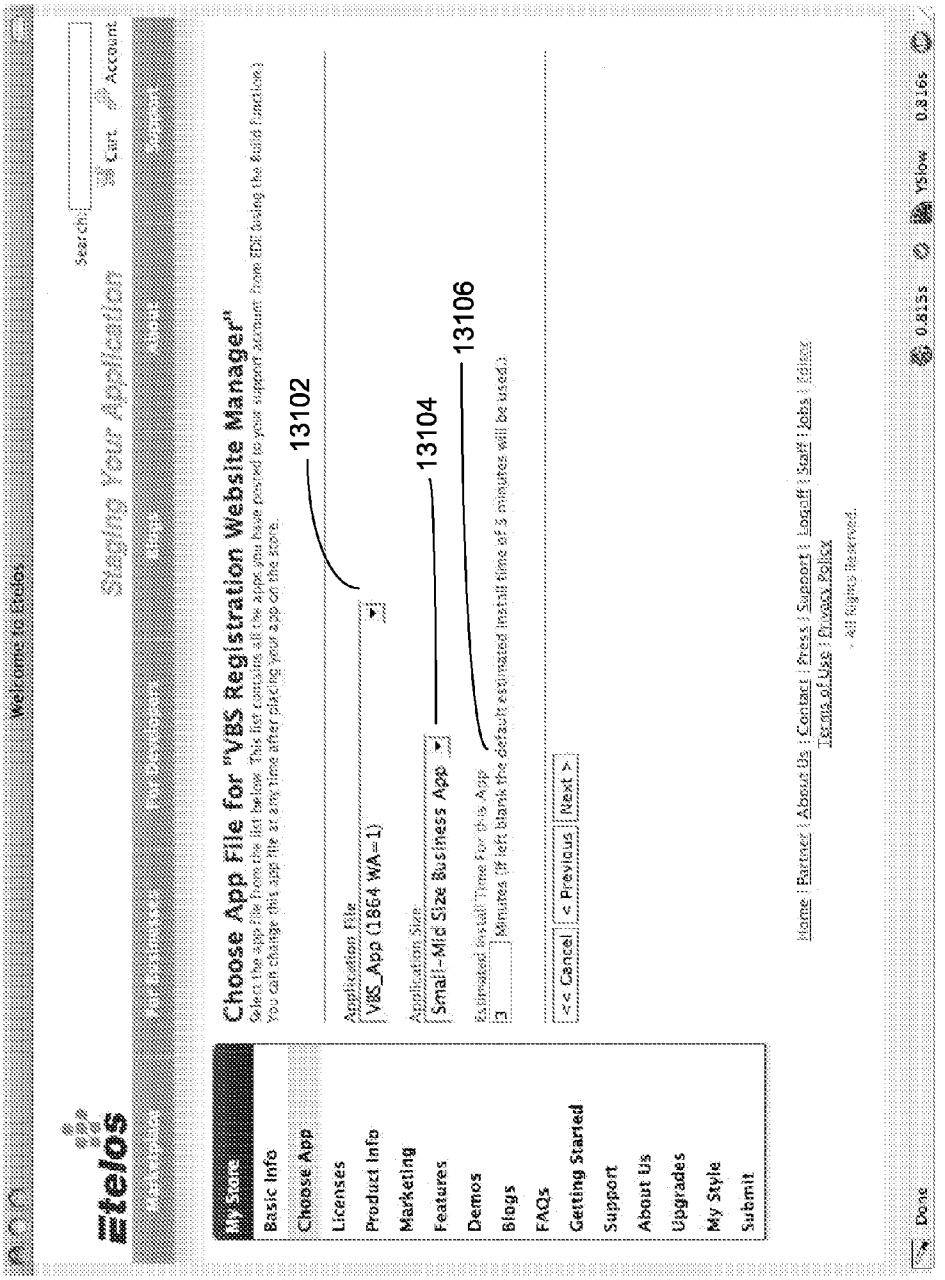

Etelos
Etelos Support
My Account Info
My Installations
Billing History
My Storefront
My Storefront
Store Reports
My Apps
My Accounts
CHP Accounts
Credit Card
Profile
My Etelos Storefront
12902
12904
12906
12908
Notification Email
update
12900

Figure 129



13000

Figure 130



13100

Figure 131

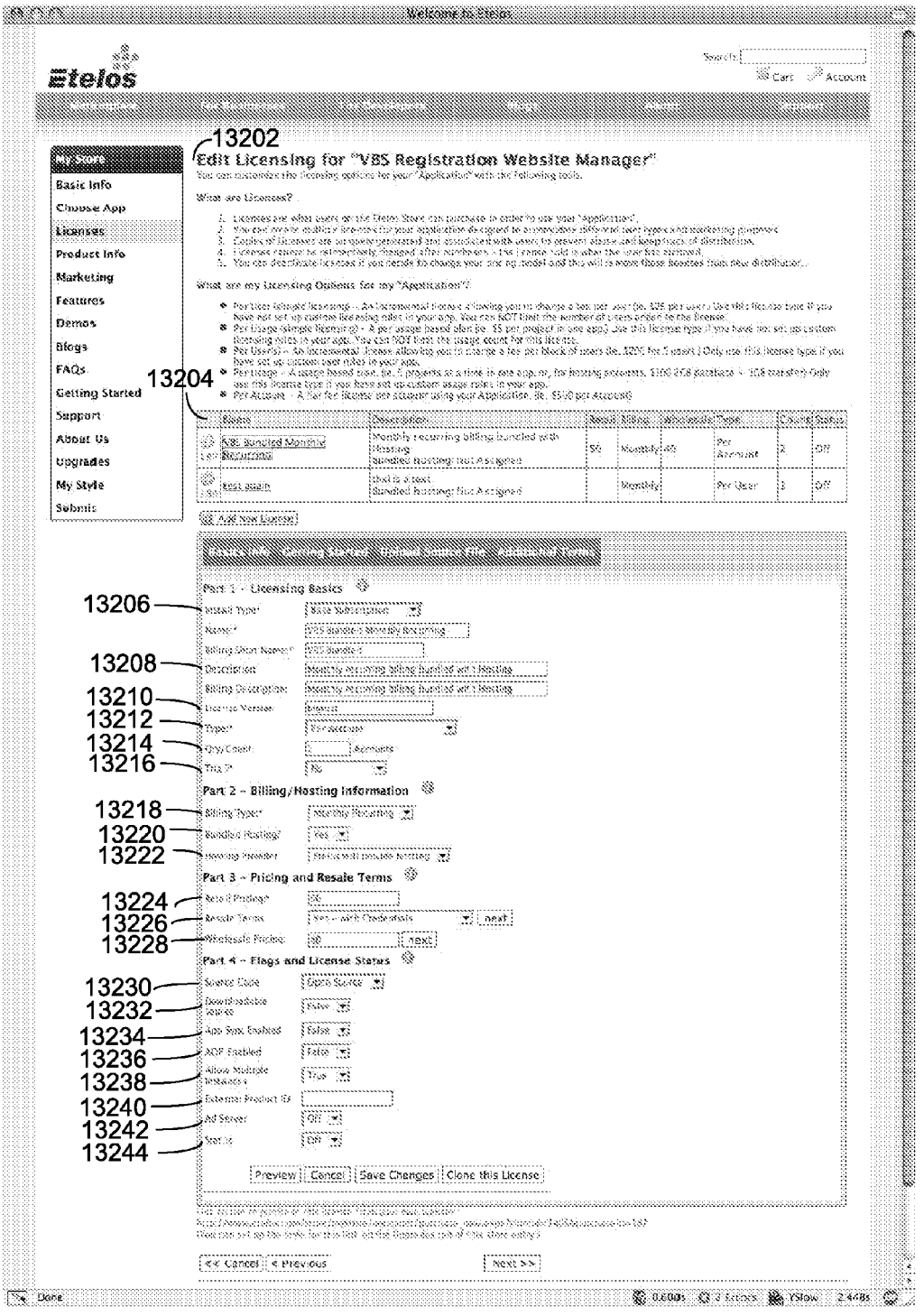

13202
Edit Licensing for "VBS Registration Website Manager"
13204
13206
13208
13210
13212
13214
13216
13218
13220
13222
13224
13226
13228
13230
13232
13234
13236
13238
13240
13242
13244
13200

Figure 132

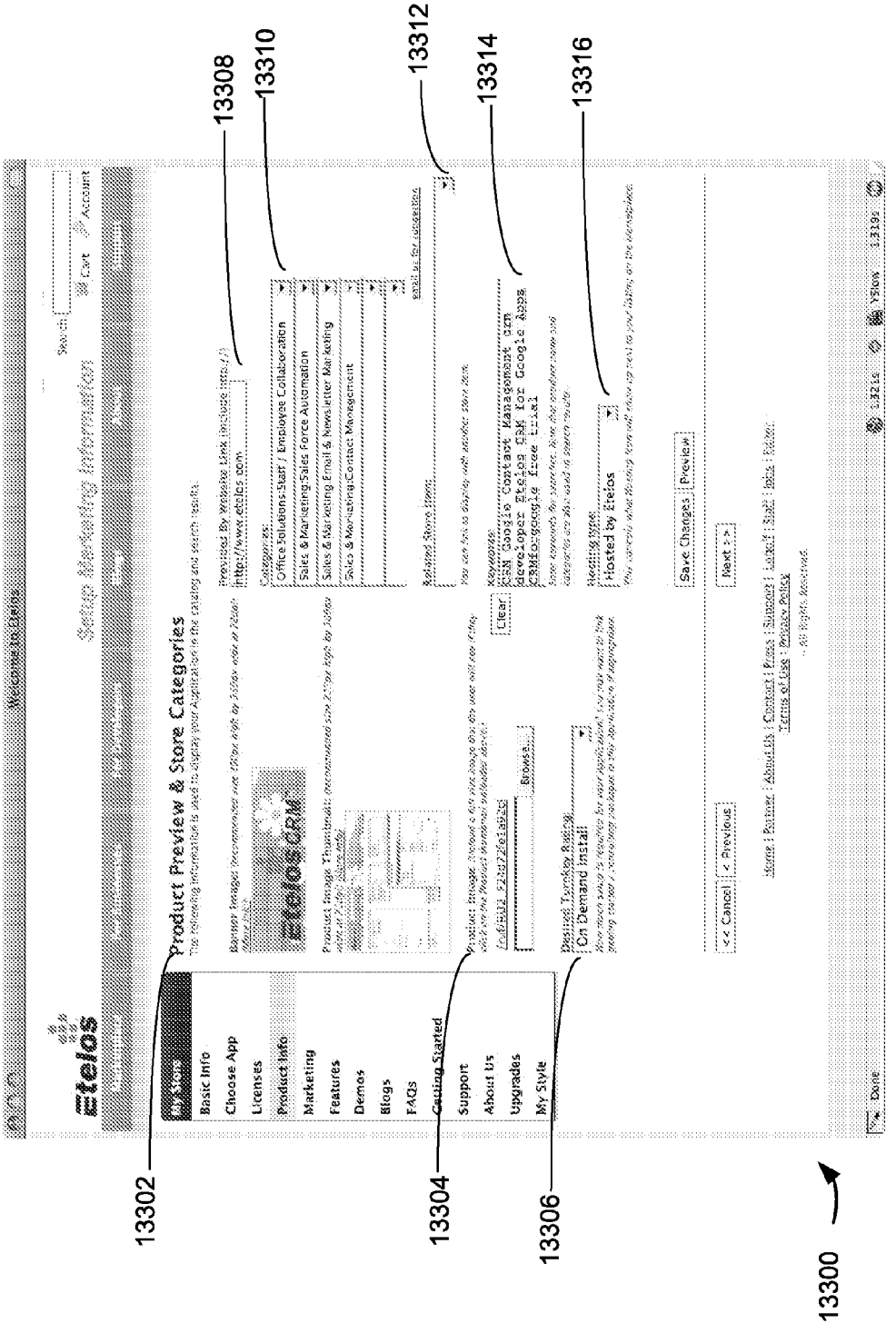

13300
13302
13304
13306
13308
13310
13312
13314
13316
Etelos
Product Preview & Store Categories
Basic Info
Choose App
Licenses
Product Info
Marketing
Features
Demos
Blogs
FAQs
Getting Started
Support
About Us
Upgrades
My Style
Save Changes
Preview
Next >>
<< Cancel
< Previous

Figure 133

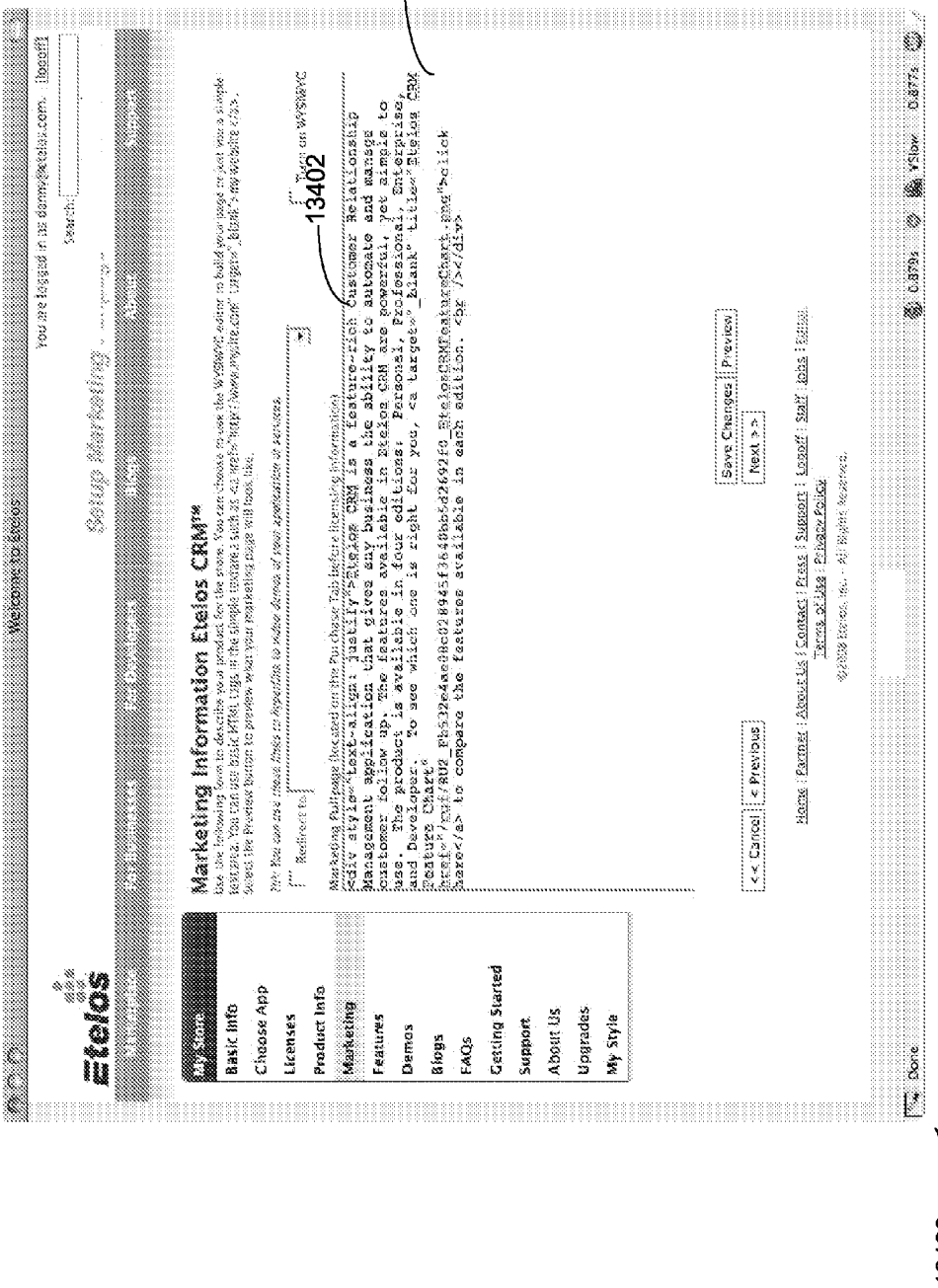

13400
13402
13404
Marketing Information Etelos CRM™
Basic Info
Choose App
Licenses
Product Info
Marketing
Features
Demos
Blogs
FAQs
Getting Started
Support
About Us
Upgrades
My Style
Save Changes
Preview
Next > >
< < Cancel
< Previous

Figure 134

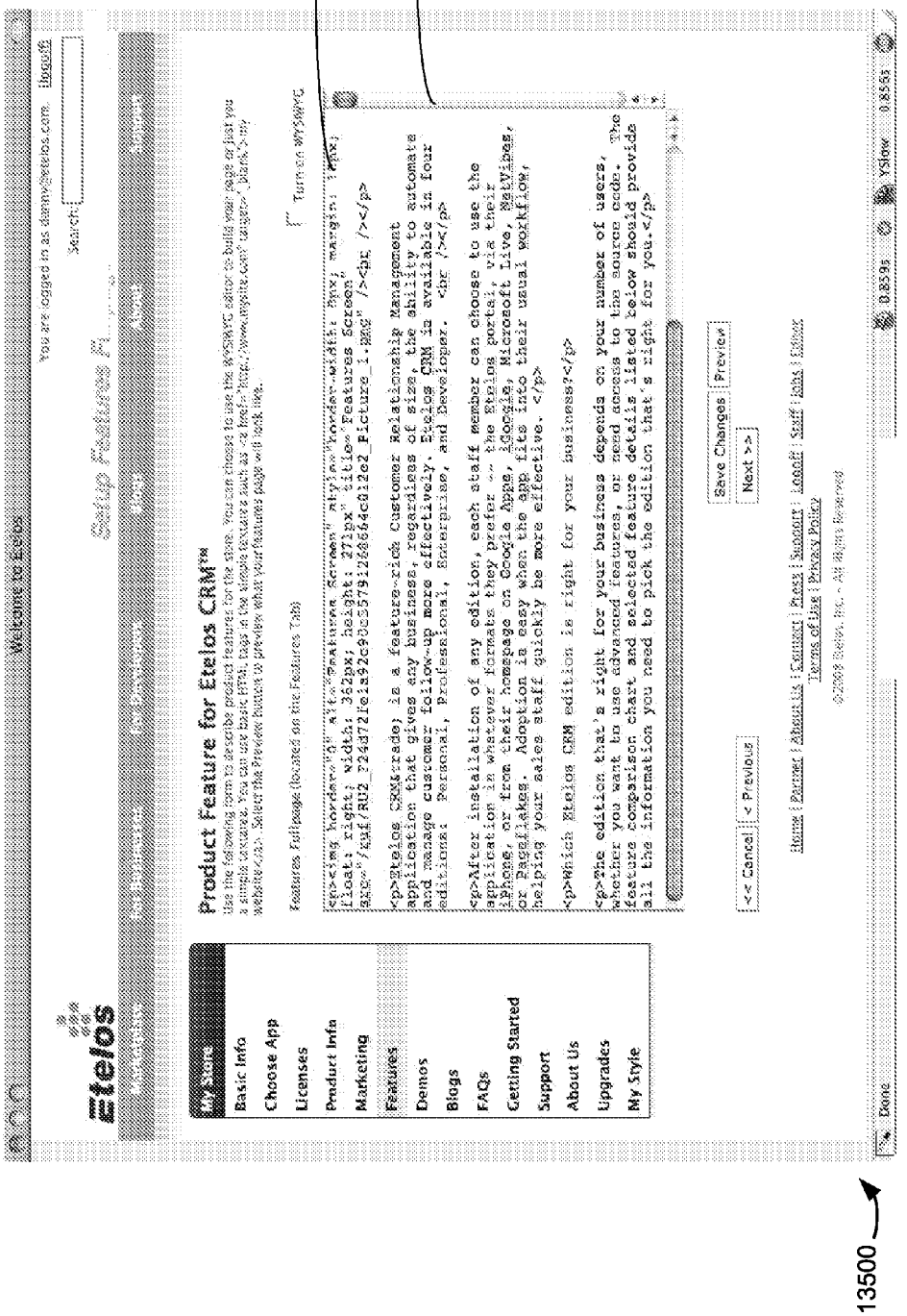

13502
13504
13500
Welcome to Etelos
Etelos
Setup Features
My Store
Basic Info
Choose App
Licenses
Product Info
Marketing
Features
Demos
Blogs
FAQs
Getting Started
Support
About Us
Upgrades
My Style
Product Feature for Etelos CRM™
Save Changes
Preview
Next >>
<< Cancel
< Previous
Done

Figure 135

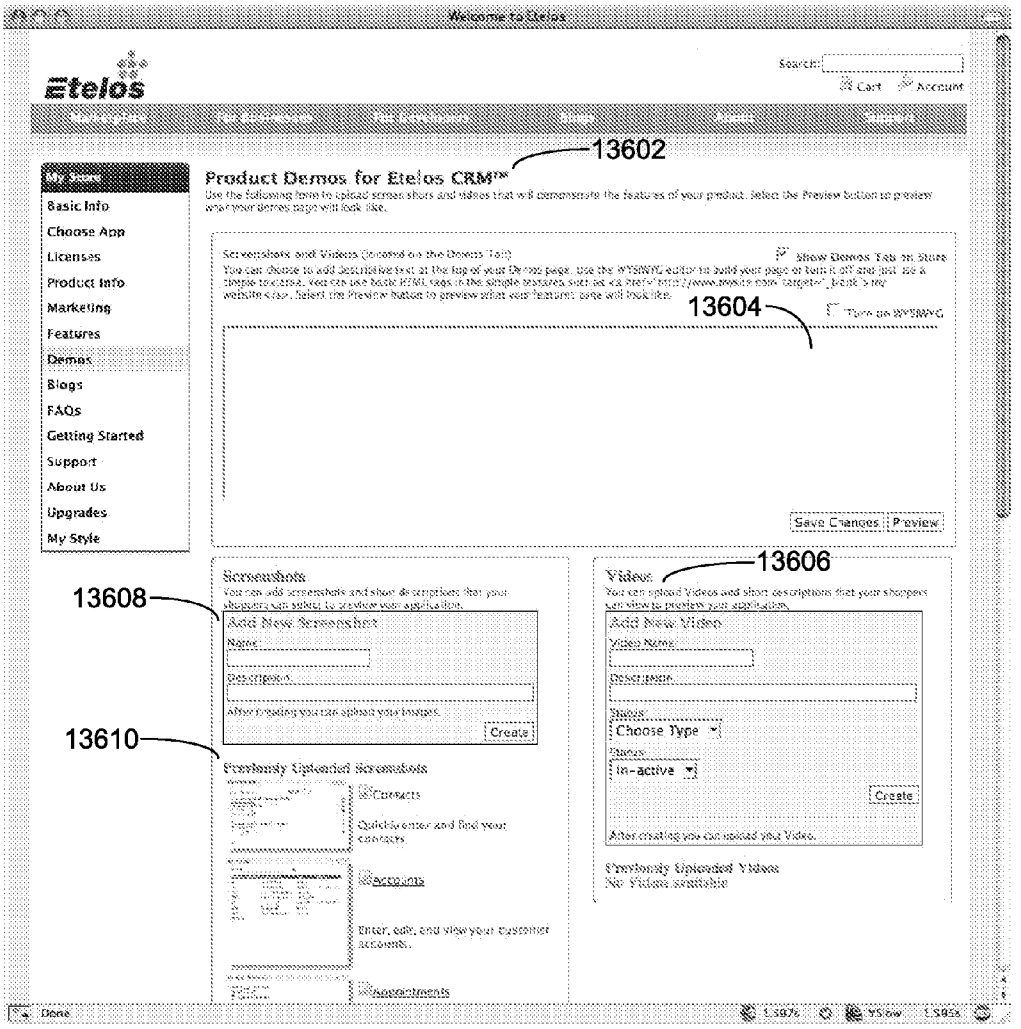

Welcome to Etelos
Etelos
Search:
Cart
Account
13602
Product Demos for Etelos CRM™
Use the following form to upload screen shots and videos that will demonstrate the features of your product. Select the Preview button to preview what your demos page will look like.
Basic Info
Choose App
Licenses
Product Info
Marketing
Features
Demos
Blogs
FAQs
Getting Started
Support
About Us
Upgrades
My Style
Screenshots and Videos (located on the Demos Tab)
Show Demos Tab on Store
13604
Save Changes
Preview
13606
Screenshots
13608
Add New Screenshot
Name:
Description:
Create
13610
Previously Uploaded Screenshots
Contacts
Quickly enter and find your contacts
Accounts
Enter, edit, and view your customer accounts.
Appointments
Video
Add New Video
Video Name:
Description:
Type:
Choose Type
Status:
In-active
Create
Previously Uploaded Videos
No Videos available
Done
13600

Figure 136

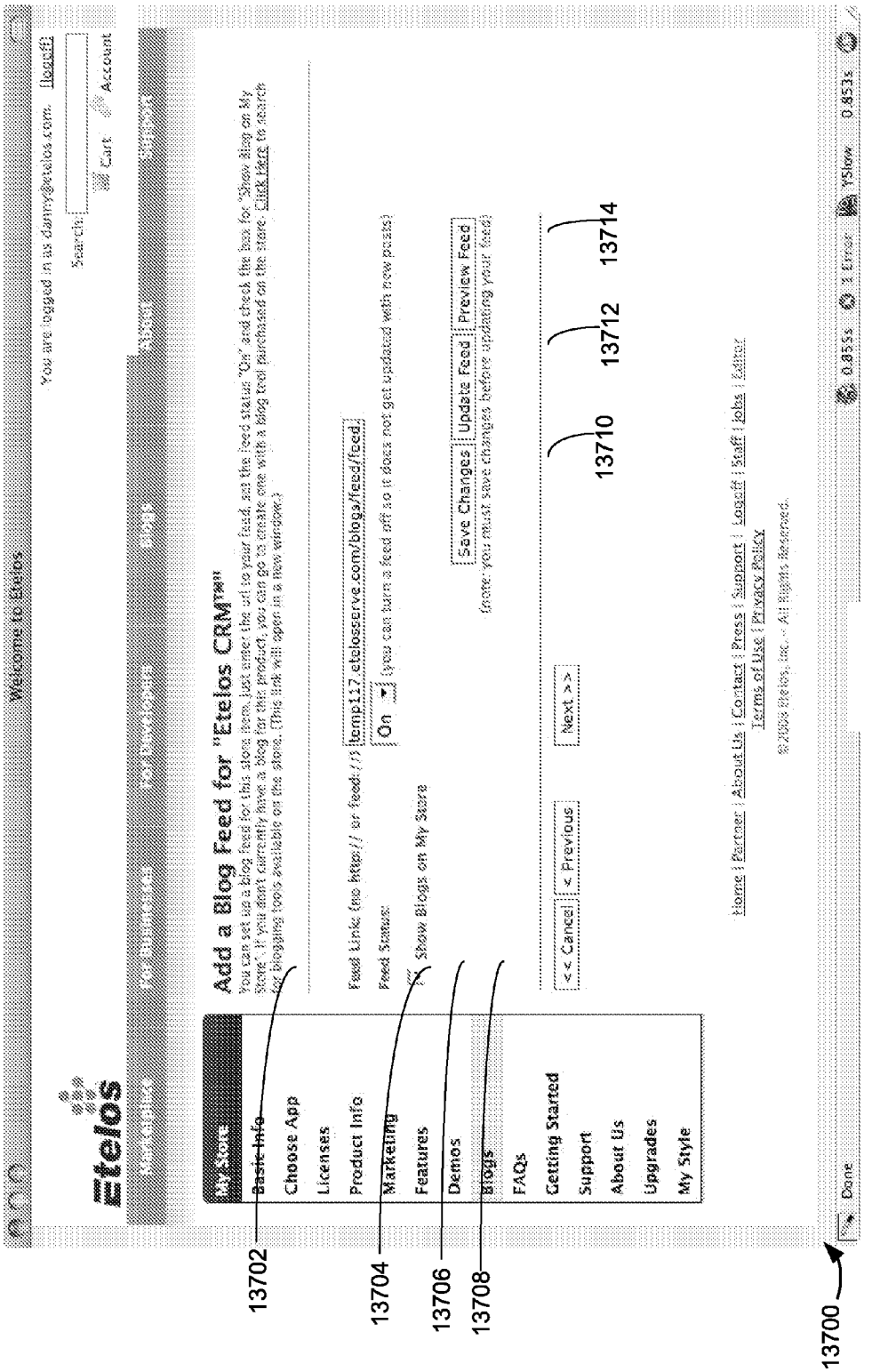

Welcome to Etelos
You are logged in as danny@etelos.com. Logoff
Search:
Cart
Account
Etelos
Add a Blog Feed for "Etelos CRM™"
You can set up a blog feed for this store item. Just enter the url to your feed, set the feed status "On" and check the box for "Show Blog on My Store". If you don't currently have a blog for the product, you can go to create one with a blog tool purchased on the store. Click Here to search for blogging tools available on the store. (This link will open in a new window.)
My Store
Basic Info
Choose App
Licenses
Product Info
Marketing
Features
Demos
Blogs
FAQs
Getting Started
Support
About Us
Upgrades
My Style
Feed Link: (no http:// or feed: /)
temp127.etelosserve.com/blogs/feed/feed
Feed Status:
On
(you can turn a feed off so it does not get updated with new posts)
Show Blogs on My Store
Save Changes
Update Feed
Preview Feed
(note: you must save changes before updating your feed)
<< Cancel
< Previous
Next >>
Home | Partner | About Us | Contact | Press | Support | Logoff | Staff | Jobs | Editor
Terms of Use | Privacy Policy
©2006 Etelos, Inc. - All Rights Reserved.
Done
0.855s
1 Error
YSlow
0.853s
13700
13702
13704
13706
13708
13710
13712
13714

Figure 137

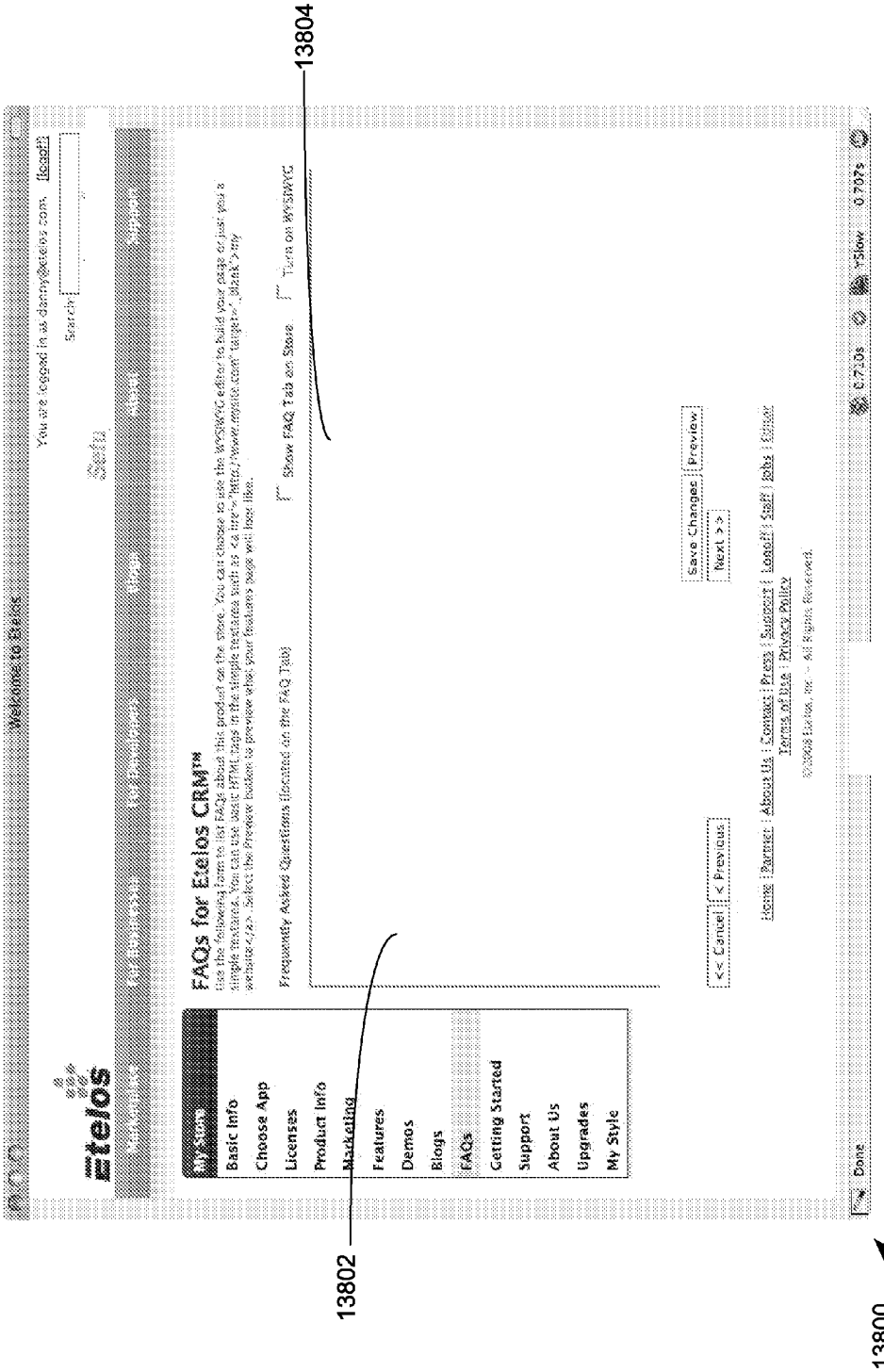

13804
13802
13800
Welcome to Etelos
FAQs for Etelos CRM™
Basic Info
Choose App
Licenses
Product Info
Marketing
Features
Demos
Blogs
FAQs
Getting Started
Support
About Us
Upgrades
My Style
Frequently Asked Questions located on the FAQ Tab
Show FAQ Tab on Store
Turn on WYSIWYG
<< Cancel
< Previous
Save Changes
Preview
Next >>

Figure 138

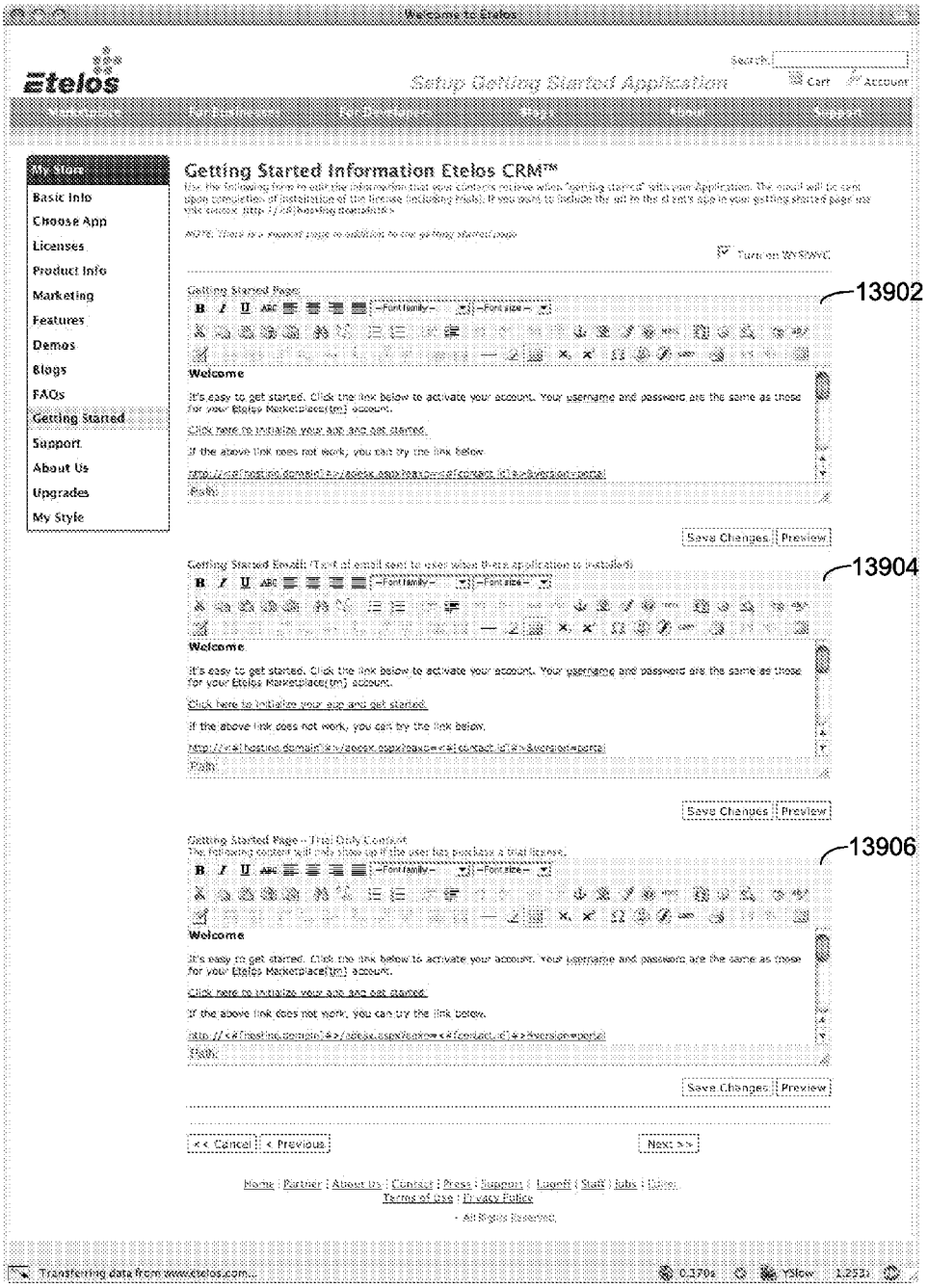

Welcome to Etelos
Etelos
Setup Getting Started Application
Search
Cart
Account
My Store
Basic Info
Choose App
Licenses
Product Info
Marketing
Features
Demos
Blogs
FAQs
Getting Started
Support
About Us
Upgrades
My Style
Getting Started Information Etelos CRM™
Turn on WYSIWYG
Getting Started Page:
13902
Welcome
It's easy to get started. Click the link below to activate your account. Your username and password are the same as those for your Etelos Marketplace(tm) account.
Click here to initialize your app and get started.
If the above link does not work, you can try the link below.
Path:
Save Changes
Preview
Getting Started Email: (Text of email sent to user when their application is installed)
13904
Welcome
It's easy to get started. Click the link below to activate your account. Your username and password are the same as those for your Etelos Marketplace(tm) account.
Click here to initialize your app and get started.
If the above link does not work, you can try the link below.
Path:
Save Changes
Preview
Getting Started Page -- Trial Only Content
The following content will only show up if the user has purchase a trial license.
13906
Welcome
It's easy to get started. Click the link below to activate your account. Your username and password are the same as those for your Etelos Marketplace(tm) account.
Click here to initialize your app and get started.
If the above link does not work, you can try the link below.
Path:
Save Changes
Preview
<< Cancel
< Previous
Next >>
Home | Partner | About Us | Contact | Press | Support | Logoff | Staff | Jobs | Other
Terms of Use | Privacy Policy
All Rights Reserved.
Transferring data from www.etelos.com...
13900

Figure 139

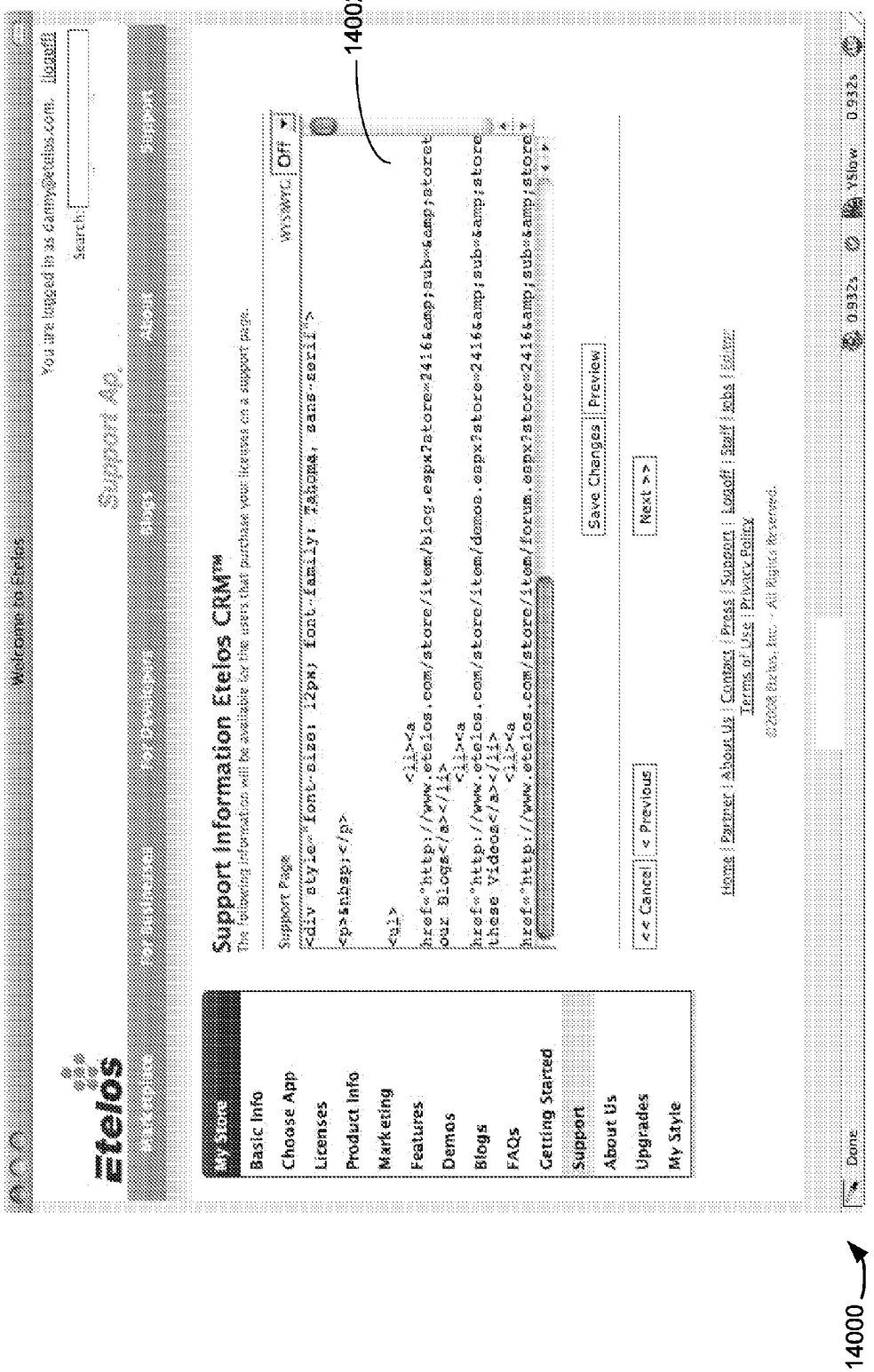

14002
14000
Welcome to Etelos
You are logged in as danny@etelos.com. logoff
Search:
Etelos
Support Ap.
My Store
Basic Info
Choose App
Licenses
Product Info
Marketing
Features
Demos
Blogs
FAQs
Getting Started
Support
About Us
Upgrades
My Style
Support Information Etelos CRM™
Support Page
Save Changes
Preview
Next >>
<< Cancel
< Previous
Done

Figure 140

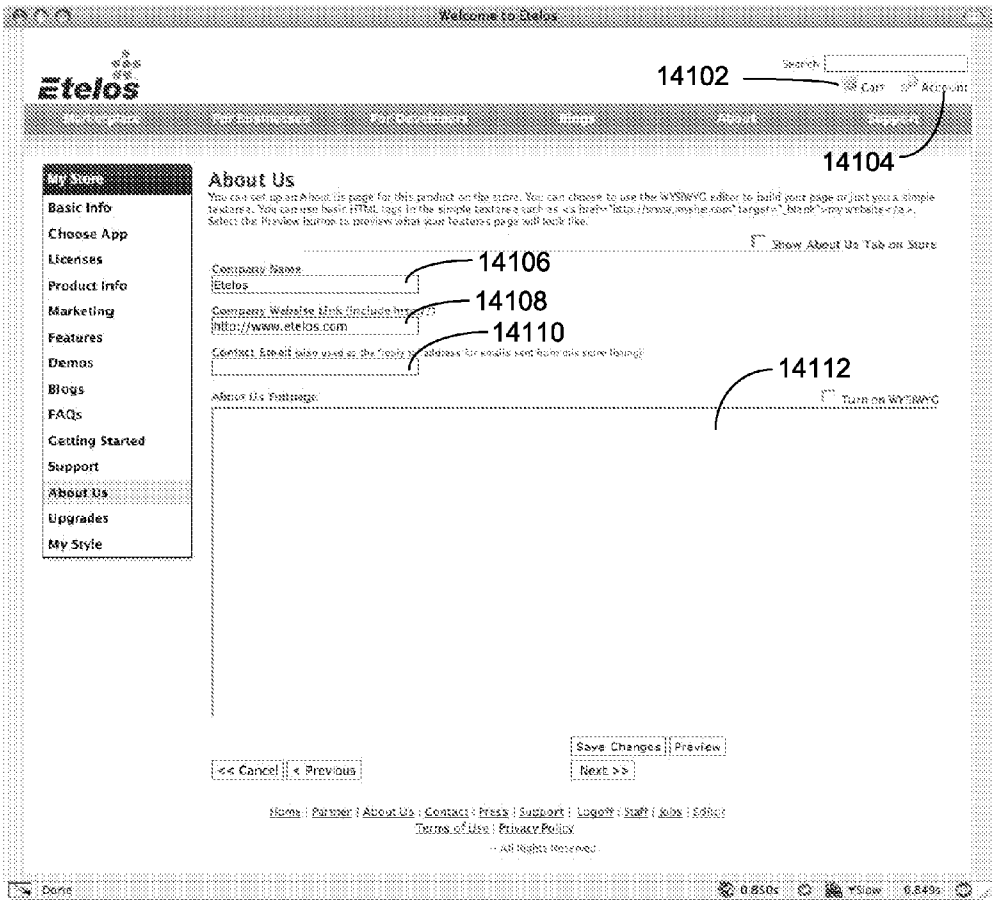

Welcome to Etelos
Etelos
14102
14104
My Store
Basic Info
Choose App
Licenses
Product Info
Marketing
Features
Demos
Blogs
FAQs
Getting Started
Support
About Us
Upgrades
My Style
About Us
14106
14108
14110
14112
Company Name
Etelos
http://www.etelos.com
Save Changes
Preview
<< Cancel
< Previous
Next >>
Done
14100

Figure 141

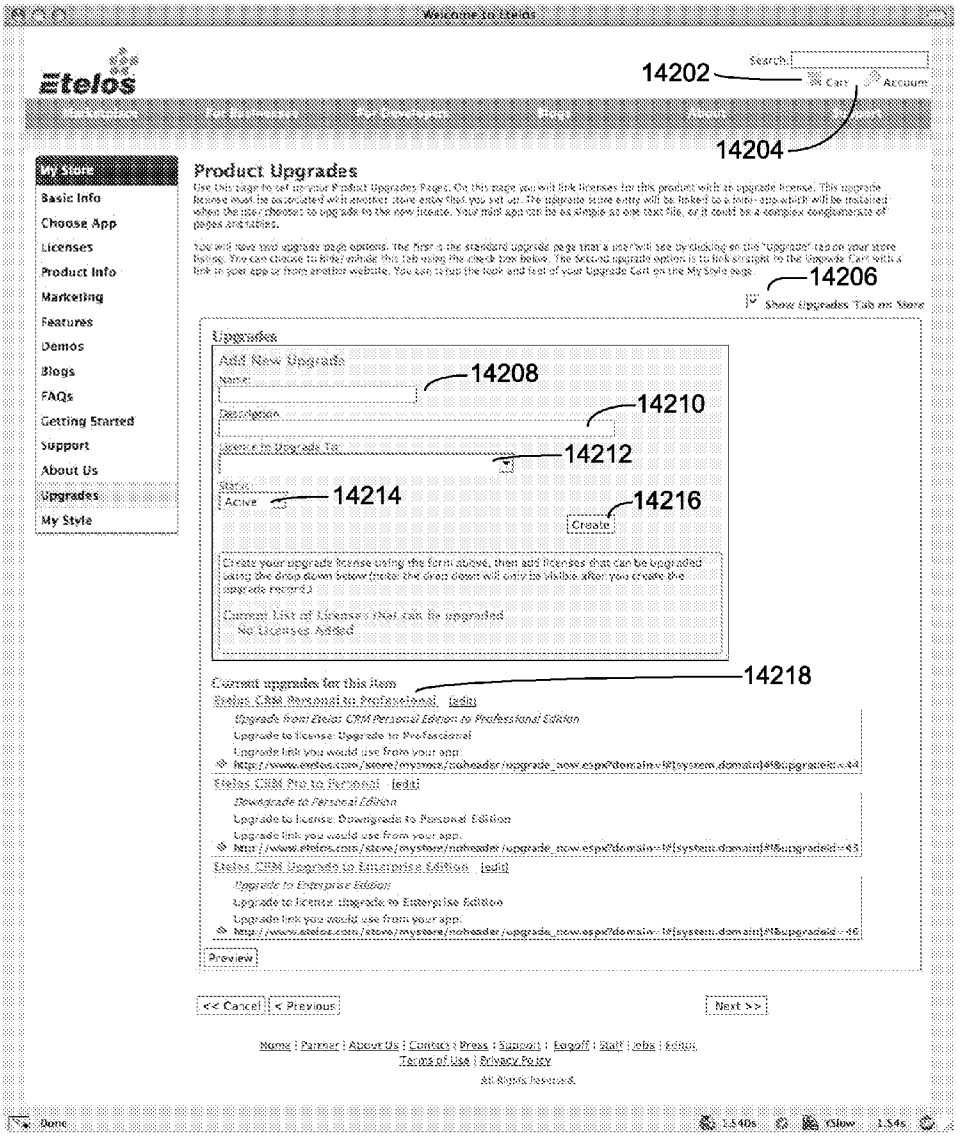

14202
14204
14206
14208
14210
14212
14214
14216
14218
14200
Product Upgrades
Upgrades
Add New Upgrade
Create
Preview
<< Cancel
< Previous
Next >>

Figure 142

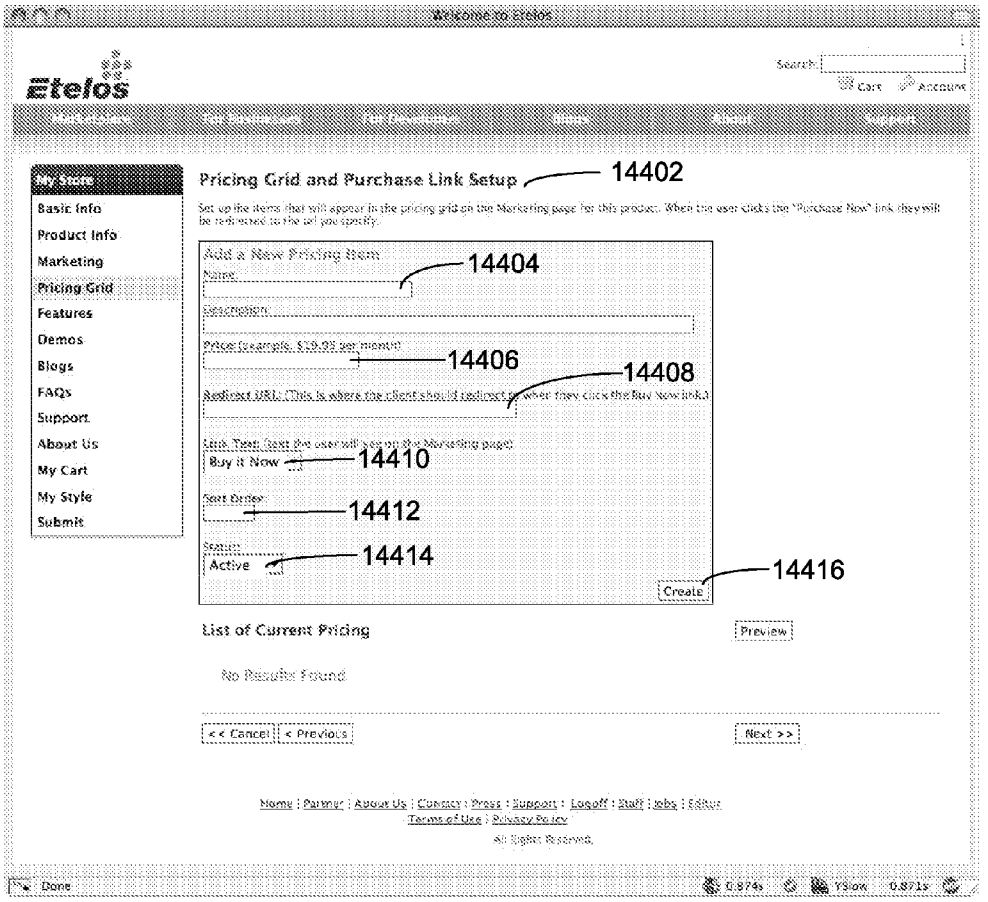
Welcome to Etelos
Etelos
Search
Cart
Account
My Store
Basic Info
Choose App
Licenses
Product Info
Marketing
Features
Demos
Blogs
FAQs
Getting Started
Support
About Us
Upgrades
My Style
My Store Style
Use this area to setup the look and feel of your store page.
Page Background Color
Choose Color
Body Font Color
#000000
Choose Color
Page Headings Font Color
#000000
Choose Color
Font
Use this style for upgrades only
14302
Page Header

<img src="http://www2.etelos.com/images/etelos-small.jpg" alt="Etelos logo"
style="padding: 15px 0px 0px 10px;">
<div style="width: 100%; height: 30px; background-image:
url(http://www.etelos.com/images/img/orangeslice.jpg); background-repeat: repeat-x;">

14304
Page Footer
<div style="border-width: 1px 0px 0px 0px; border-style: solid; border-color:
#F96600;">

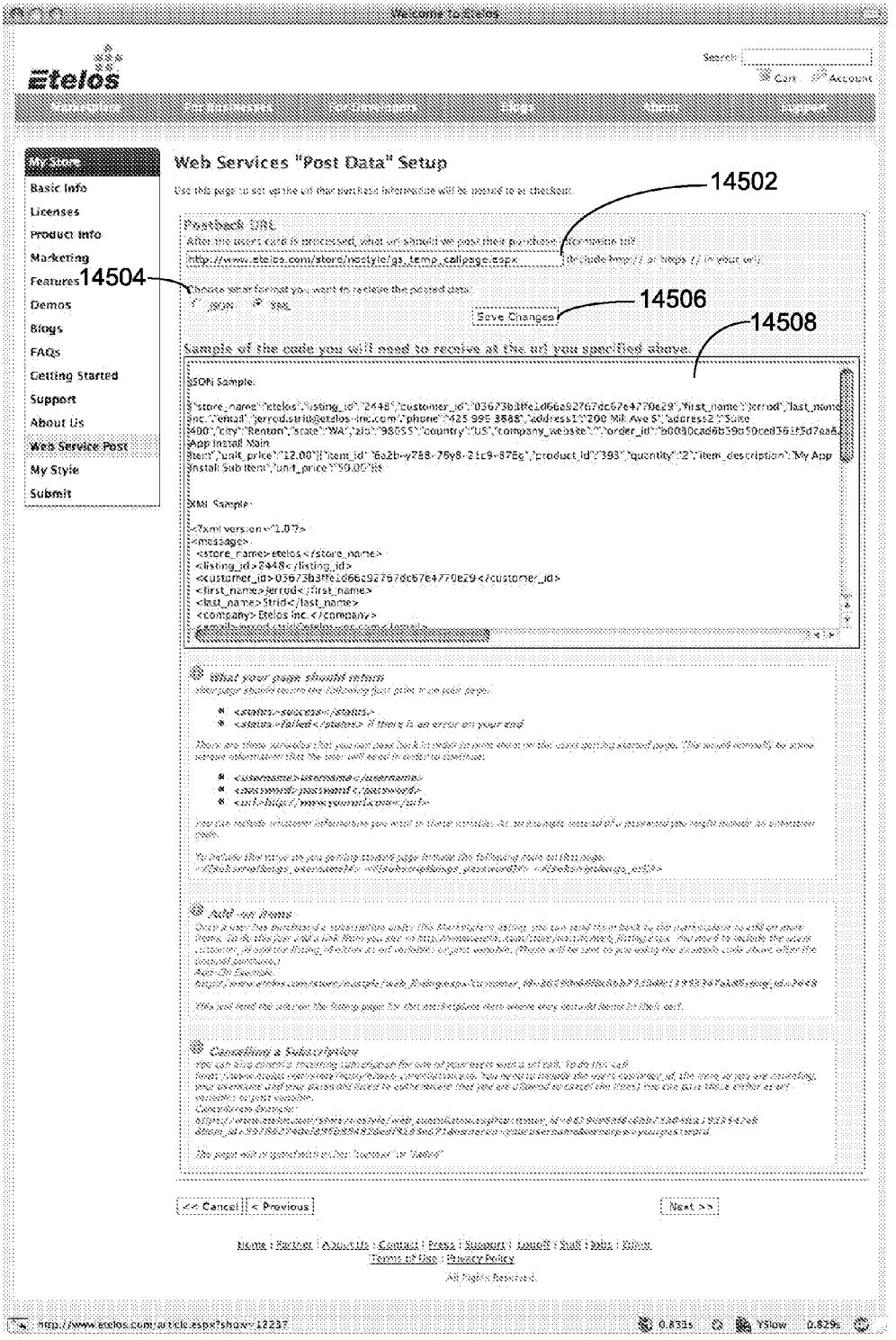
14306
Save Changes
Preview Cart
<< Cancel
< Previous
Next >>
Home | Partner | About Us | Contact | Press | Support | Logoff | Staff | Jobs | Sales
Terms of Use | Privacy Policy
All Rights Reserved.
14300

Figure 143



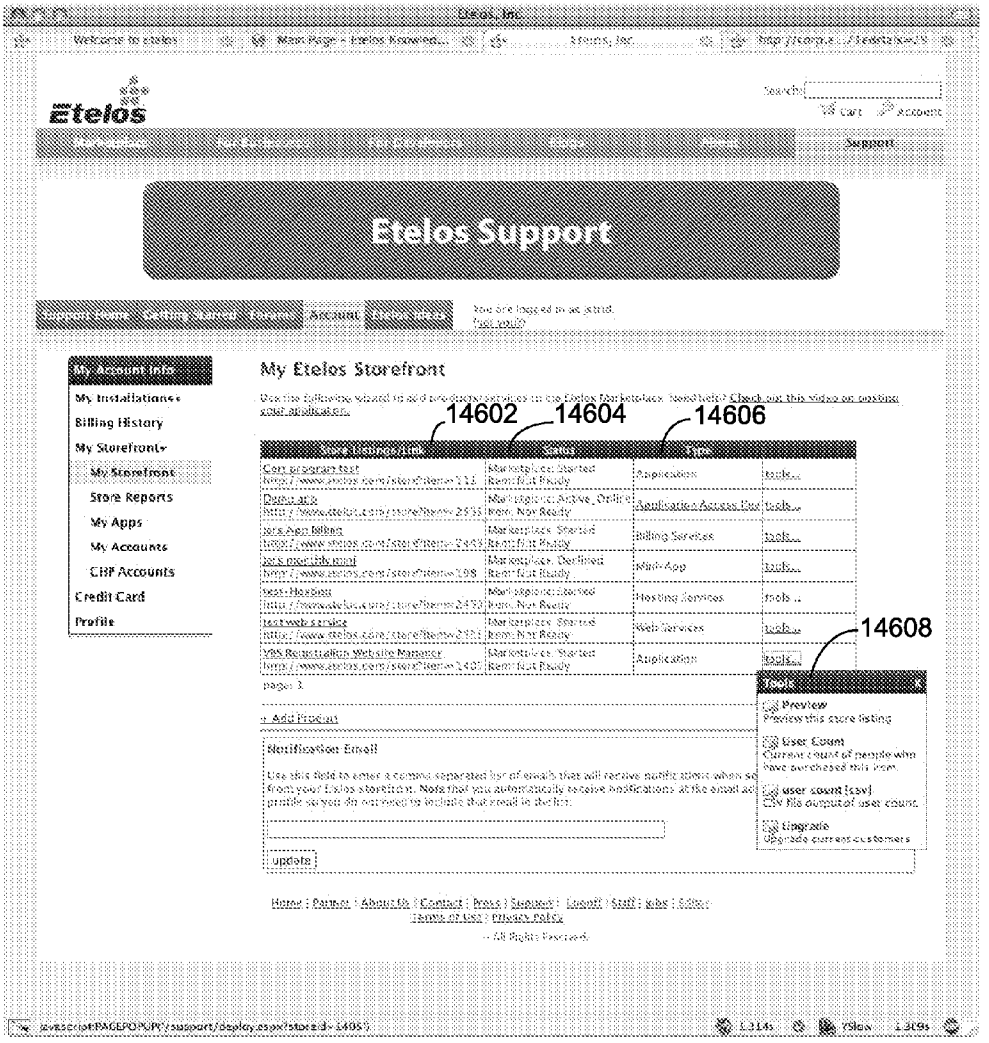
14400

Figure 144



14500

Figure 145



14600

Figure 146

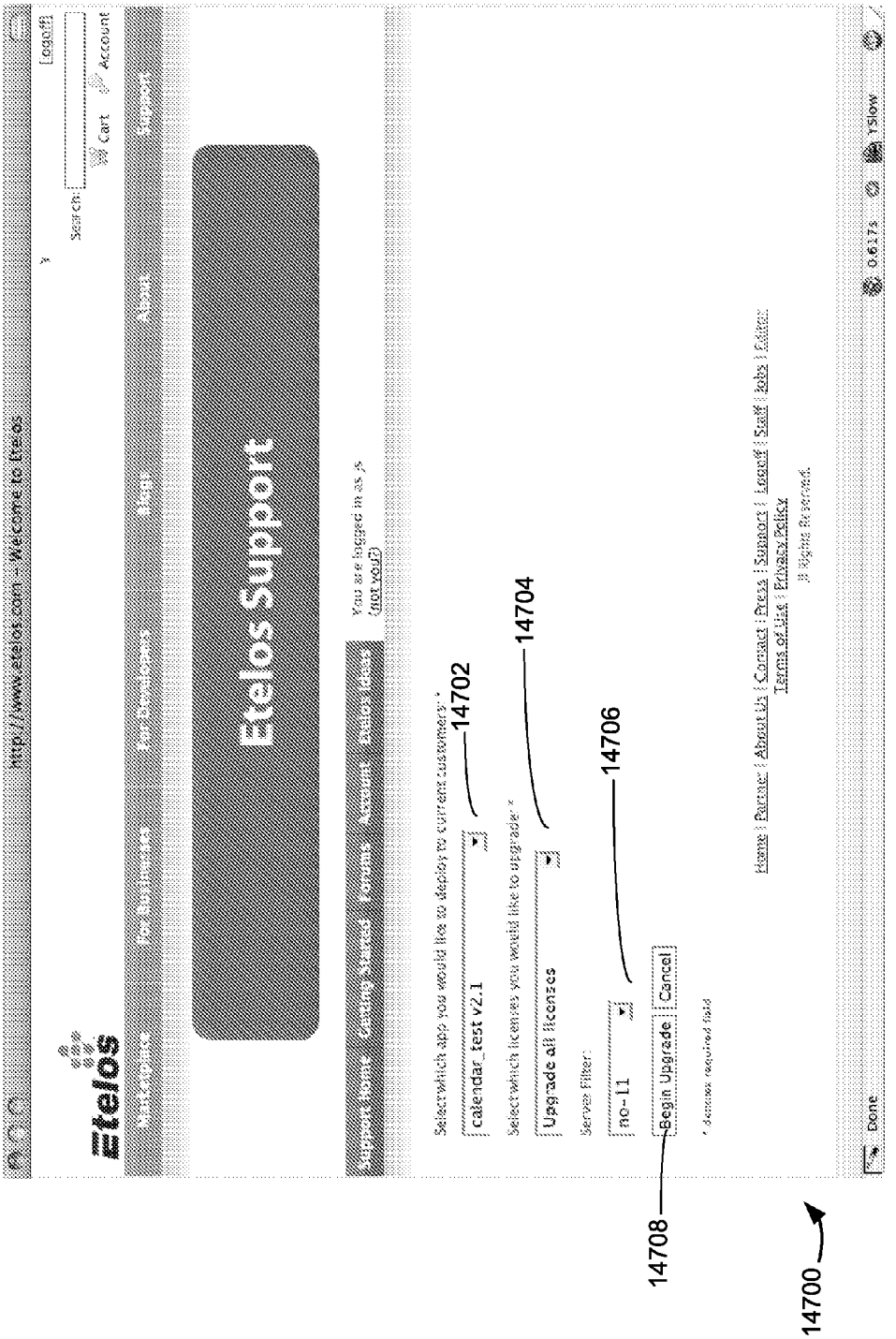

http://www.etelos.com - Welcome to Etelos
Etelos
Etelos Support
Support Home
Getting Started
Forums
Account
Etelos Ideas
You are logged in as js
(not you?)
Select which app you would like to deploy to current customers: *
calendar_test v2.1
Select which licenses you would like to upgrade: *
Upgrade all licenses
Server Filter:
no-11
Begin Upgrade
Cancel
* denotes required field
Home | Partner | About Us | Contact | Press | Support | Logoff | Staff | Jobs | Editor
Terms of Use | Privacy Policy
Done
14700
14702
14704
14706
14708

Figure 147

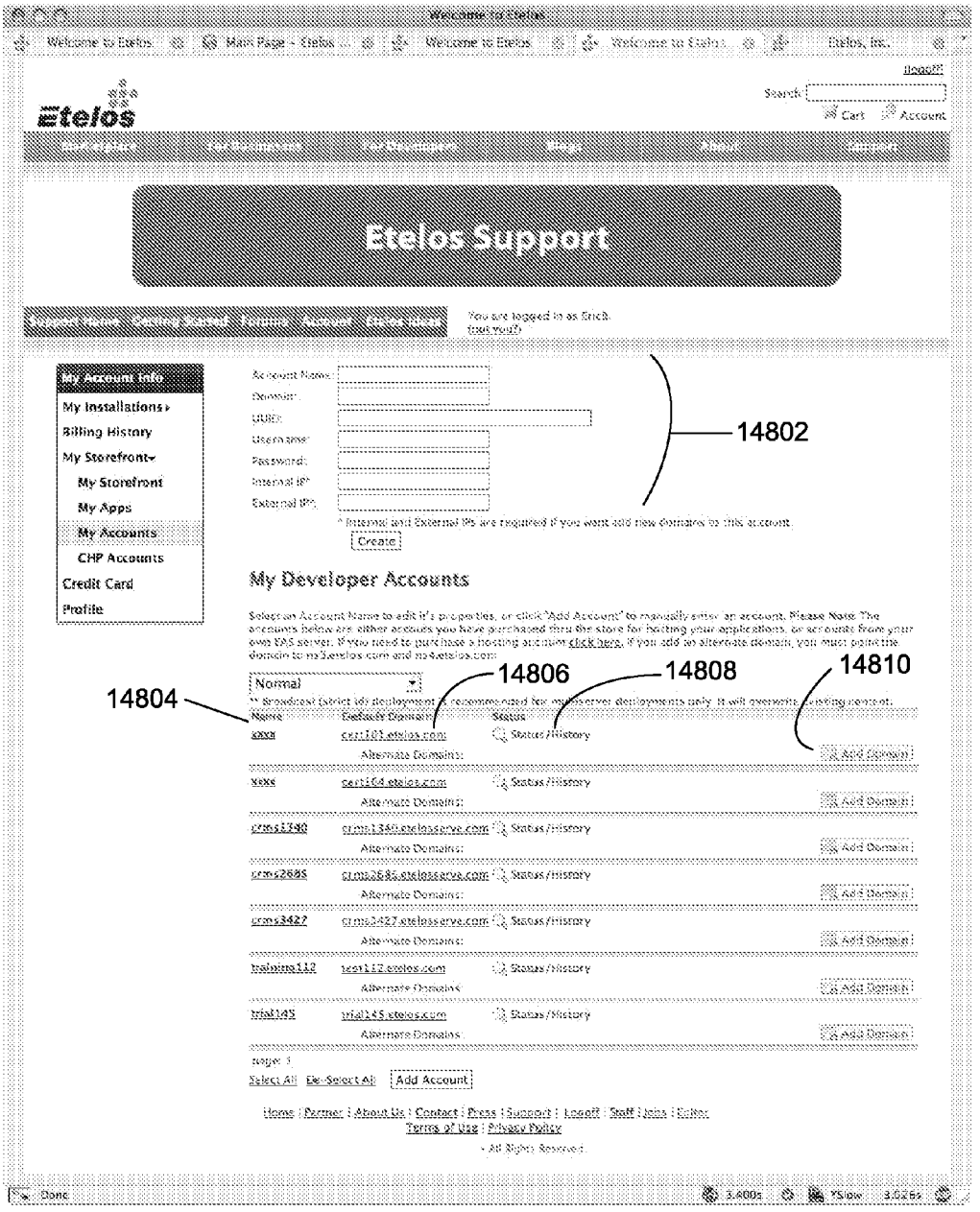

Etelos
Etelos Support
My Account Info
My Installations
Billing History
My Storefront
My Storefront
My Apps
My Accounts
CHP Accounts
Credit Card
Profile
14802
My Developer Accounts
14804
14806
14808
14810
Add Account
14800

Figure 148

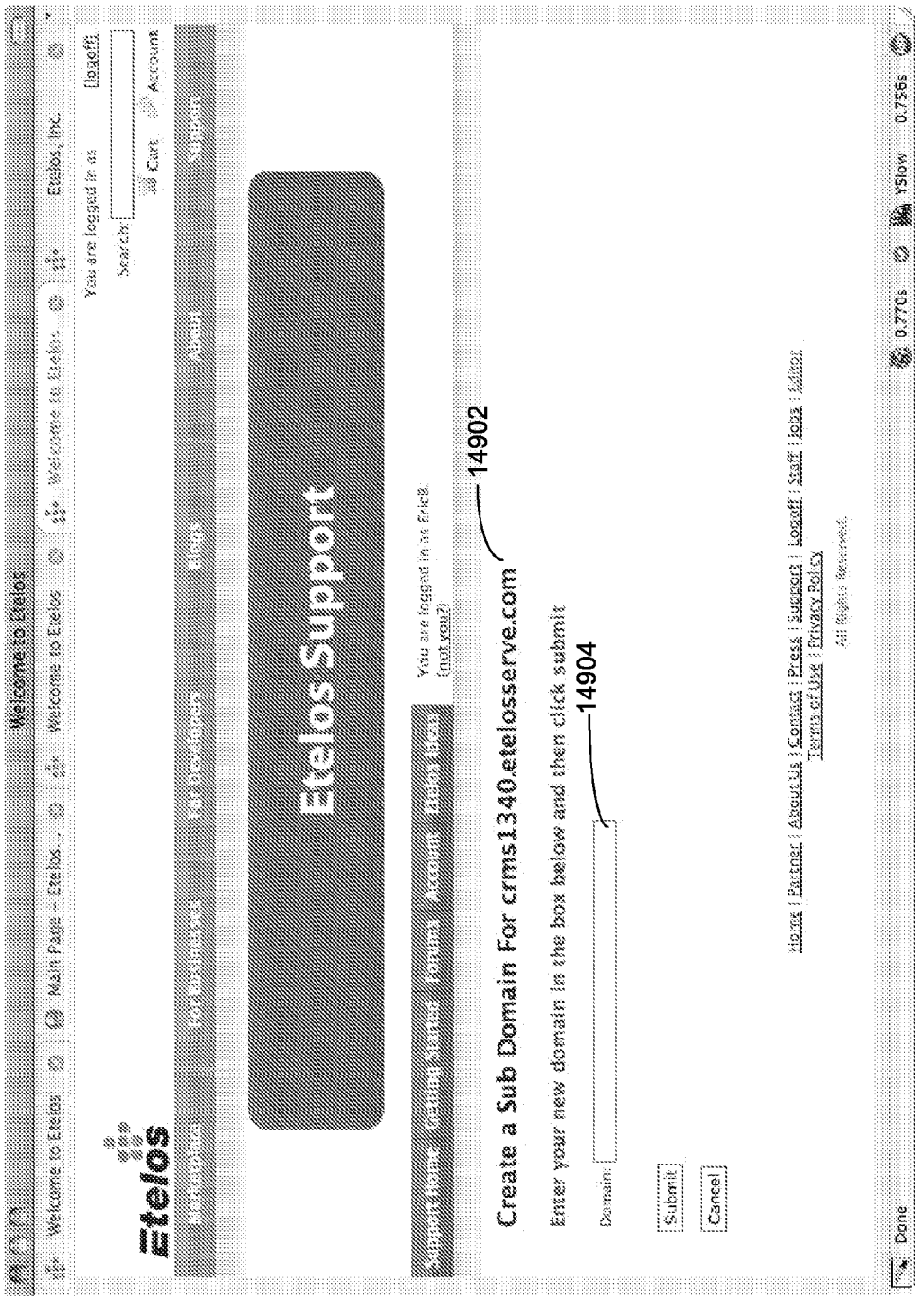
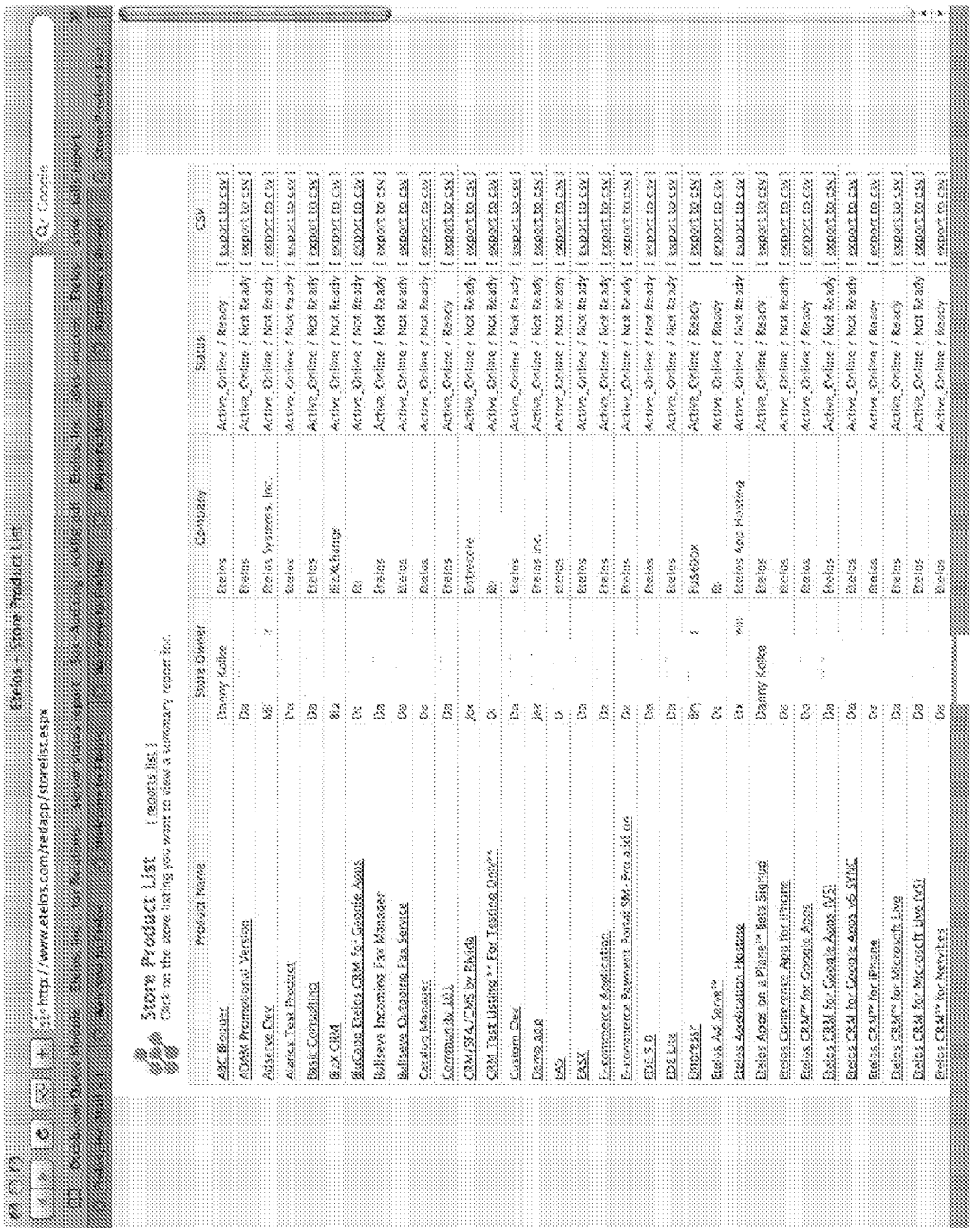

Welcome to Etelos
Etelos
Etelos Support
Create a Sub Domain For crms1340.etelosserve.com
Enter your new domain in the box below and then click submit
Domain:
Submit
Cancel
14902
14904
14900

Figure 149



15000

Figure 150

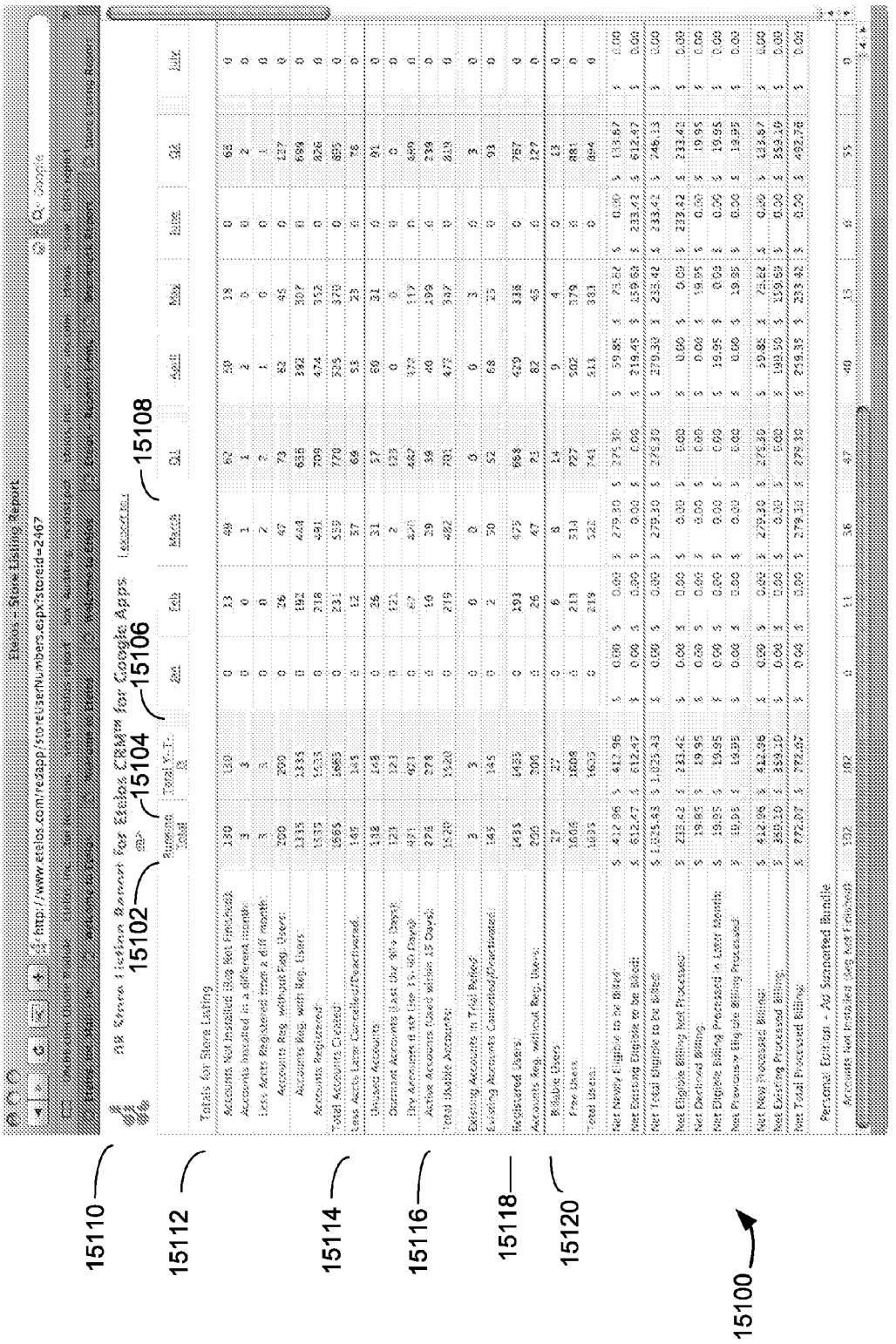

15110
15112
15114
15116
15118
15120
15100
15102
15104
15106
15108

Figure 151

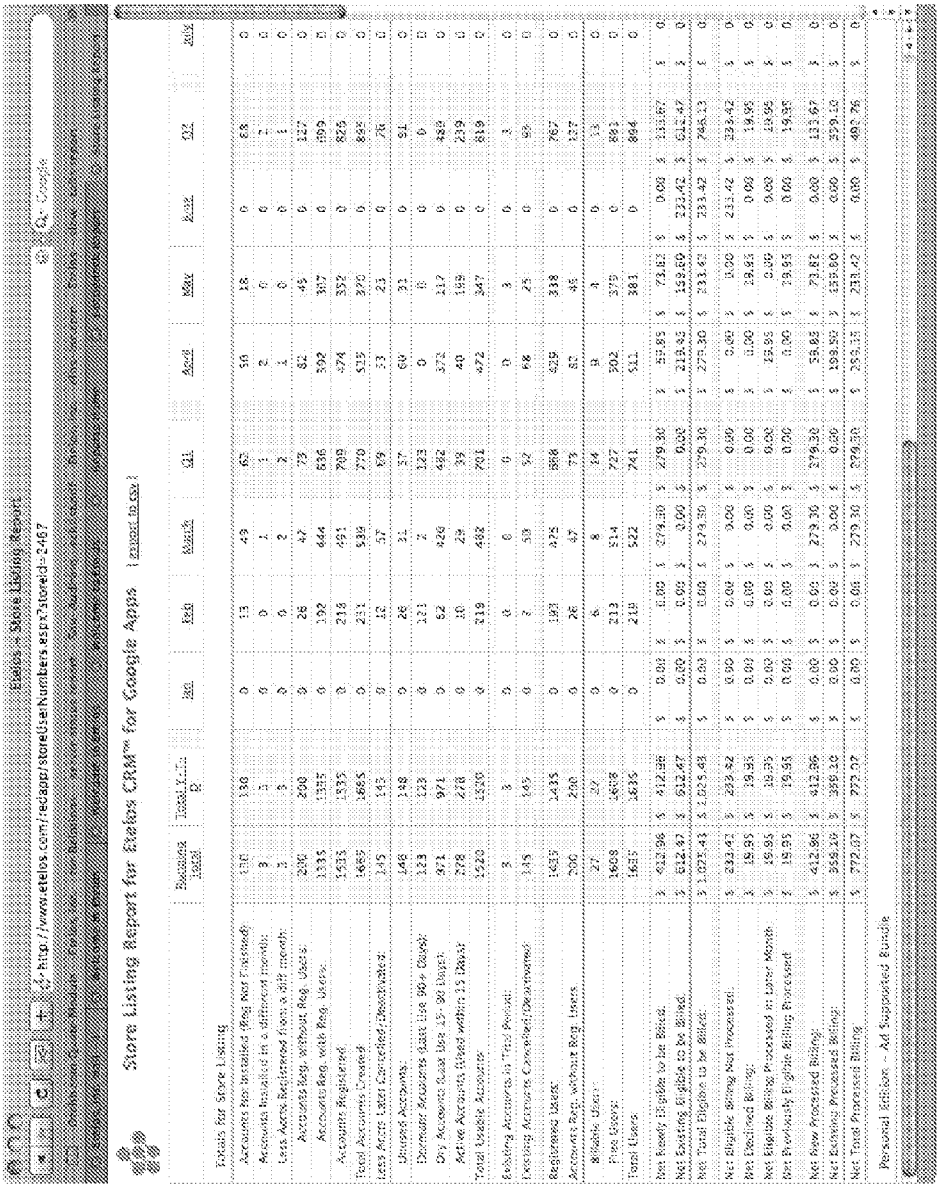


Figure 152

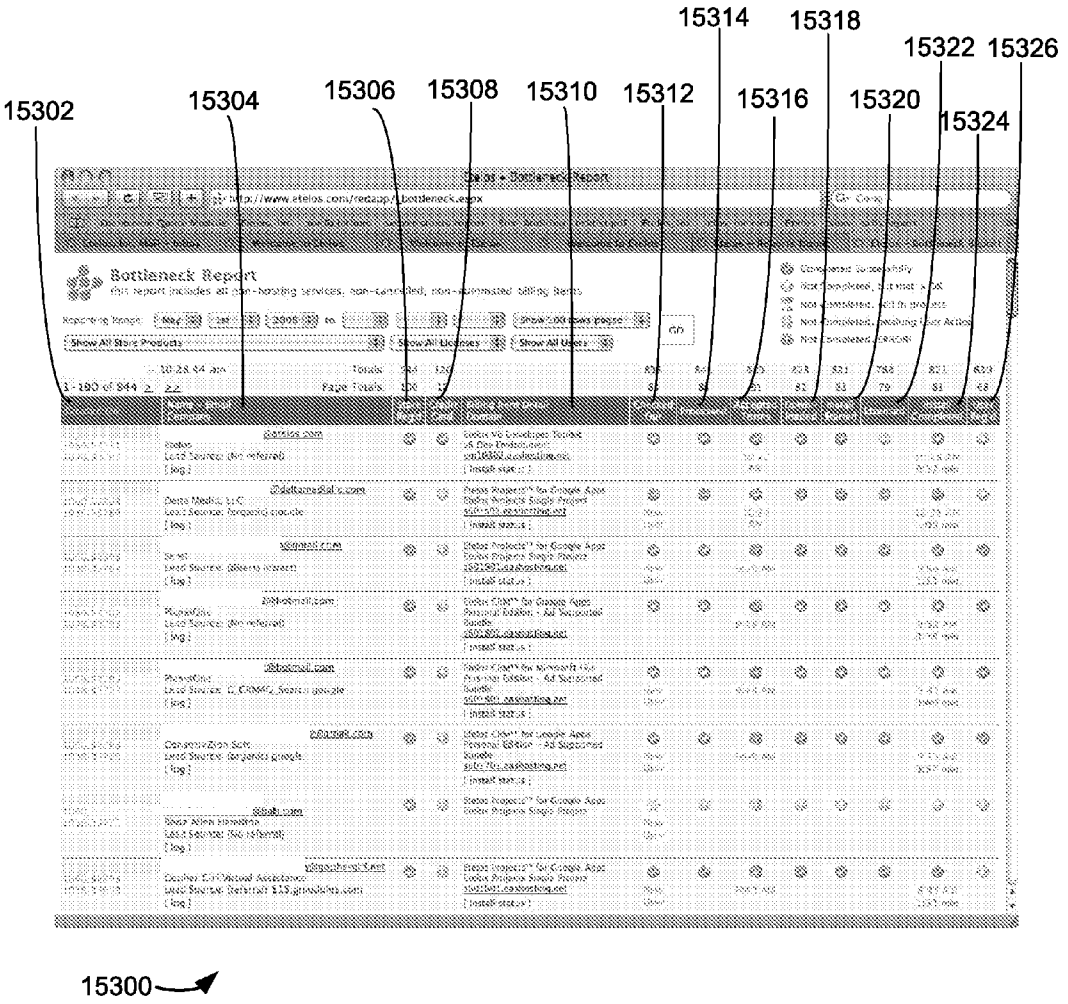

15302
15304
15306
15308
15310
15312
15314
15316
15318
15320
15322
15324
15326
Bottleneck Report
15300

Figure 153

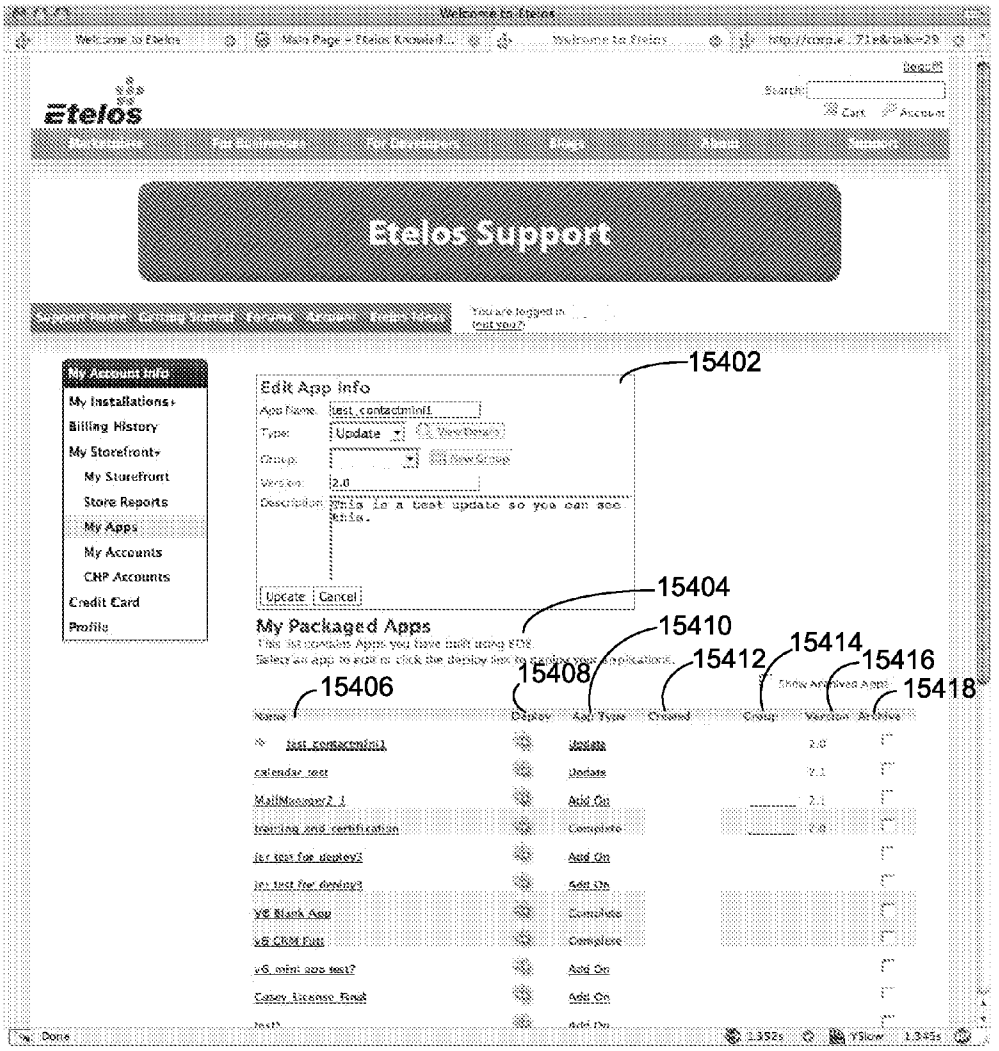

Etelos
Etelos Support
My Account Info
My Installations
Billing History
My Storefront
My Storefront
Store Reports
My Apps
My Accounts
CNP Accounts
Credit Card
Profile
Edit App Info
Update
Cancel
My Packaged Apps
15400
15402
15404
15406
15408
15410
15412
15414
15416
15418

Figure 154

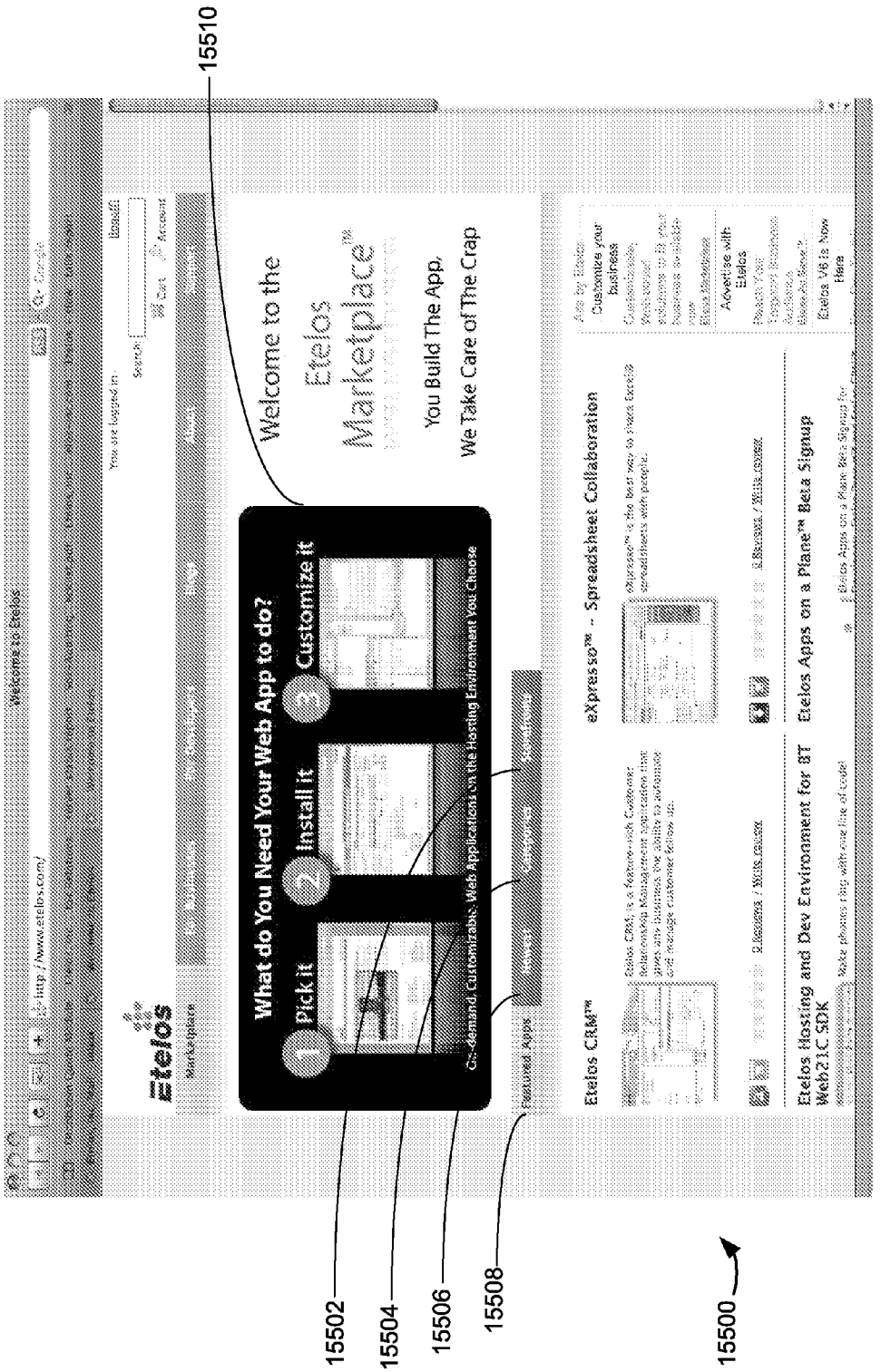

15510
Welcome to the
Etelos
Marketplace™
You Build The App.
We Take Care of The Crap
What do You Need Your Web App to do?
1 Pick it
2 Install it
3 Customize it
On-demand, Customizable, Web Applications on the Hosting Environment You Choose
Etelos CRM™
eXpresso™ - Spreadsheet Collaboration
Etelos Hosting and Dev Environment for BT Web2.0 SDK
Etelos Apps on a Plane™ Beta Signup
15502
15504
15506
15508
15500

Figure 155

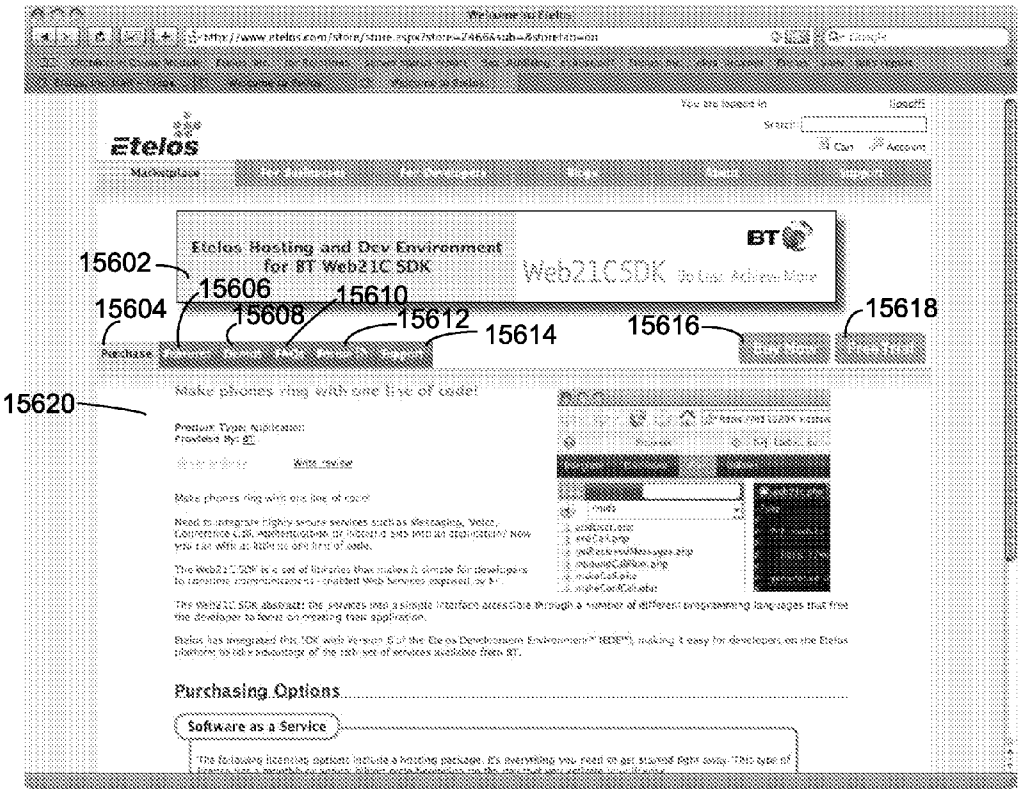

15602
15604
15606
15608
15610
15612
15614
15616
15618
15620
15600
Etelos
Etelos Hosting and Dev Environment for BT Web21C SDK
Web21CSDK
Purchase
Make phones ring with one line of code!
Purchasing Options
Software as a Service

Figure 156

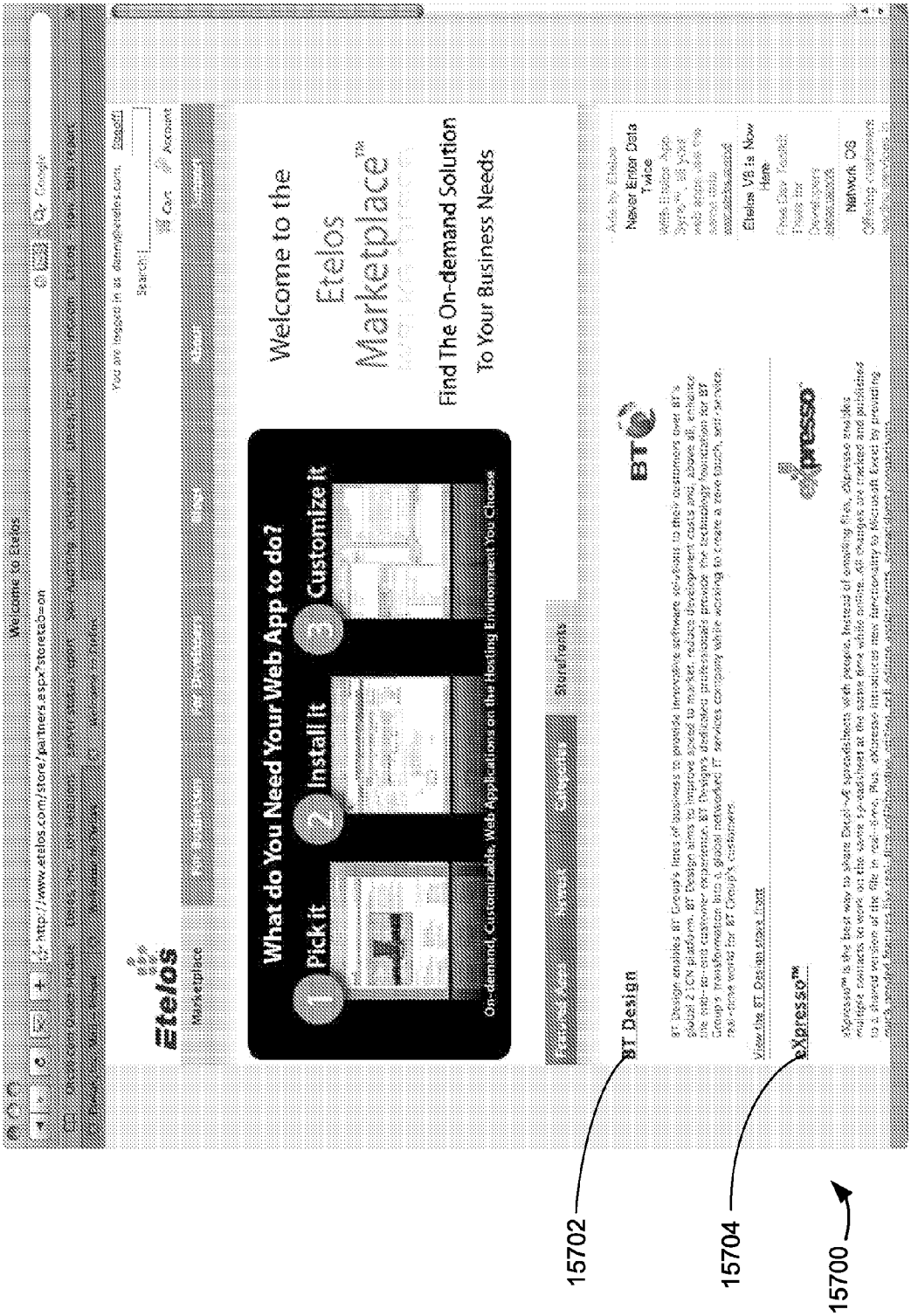

Welcome to the
Etelos
Marketplace™
Find The On-demand Solution
To Your Business Needs
What do You Need Your Web App to do?
1 Pick it
2 Install it
3 Customize It
On-demand, Customizable, Web Applications on the Hosting Environment You Choose
BT Design
eXpresso™
15702
15704
15700

Figure 157

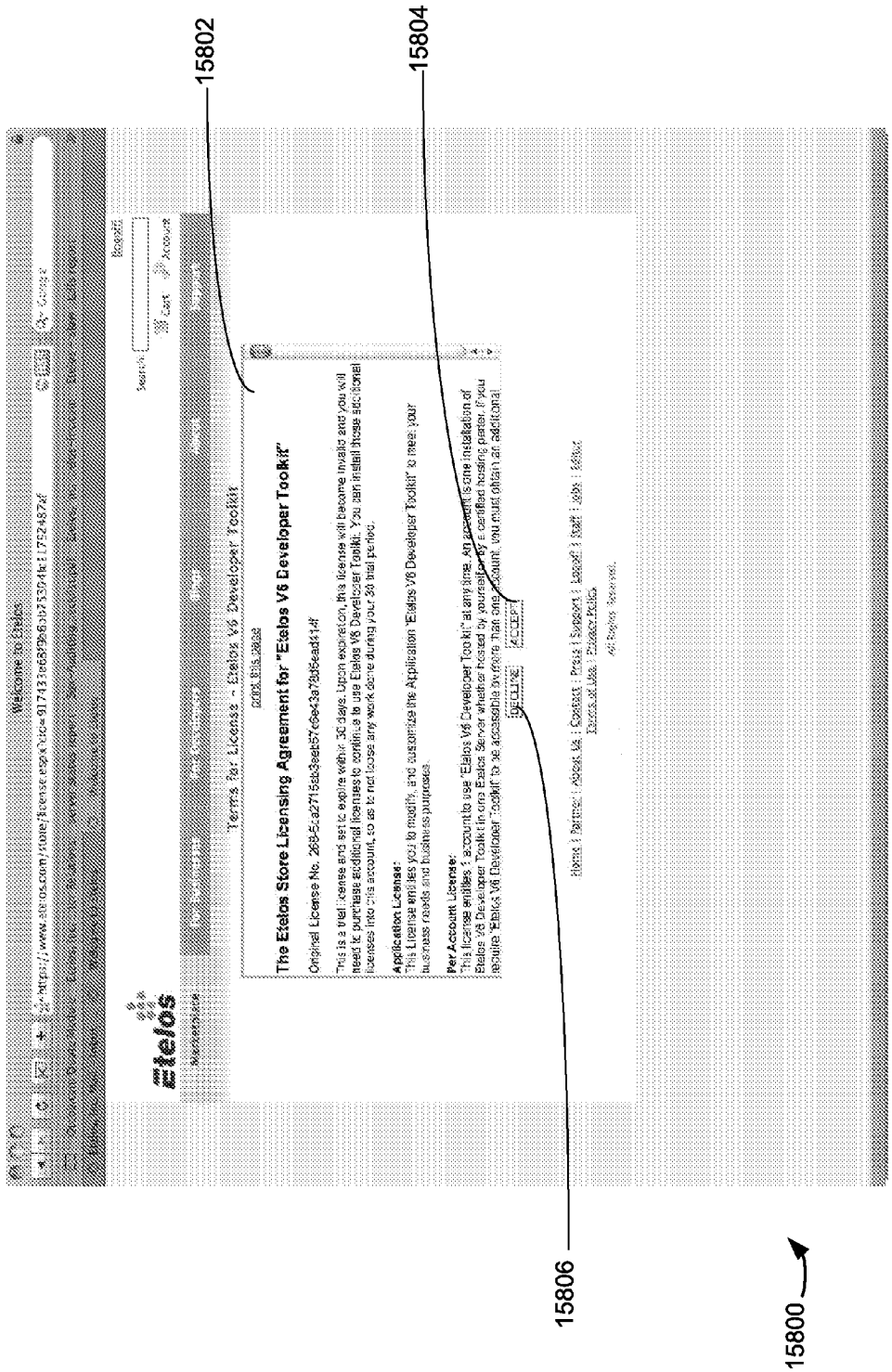

15802
15804
15806
15800
Welcome to Etelos
Etelos
Terms for License - Etelos V6 Developer Toolkit
The Etelos Store Licensing Agreement for "Etelos V6 Developer Toolkit"
DECLINE
ACCEPT

Figure 158

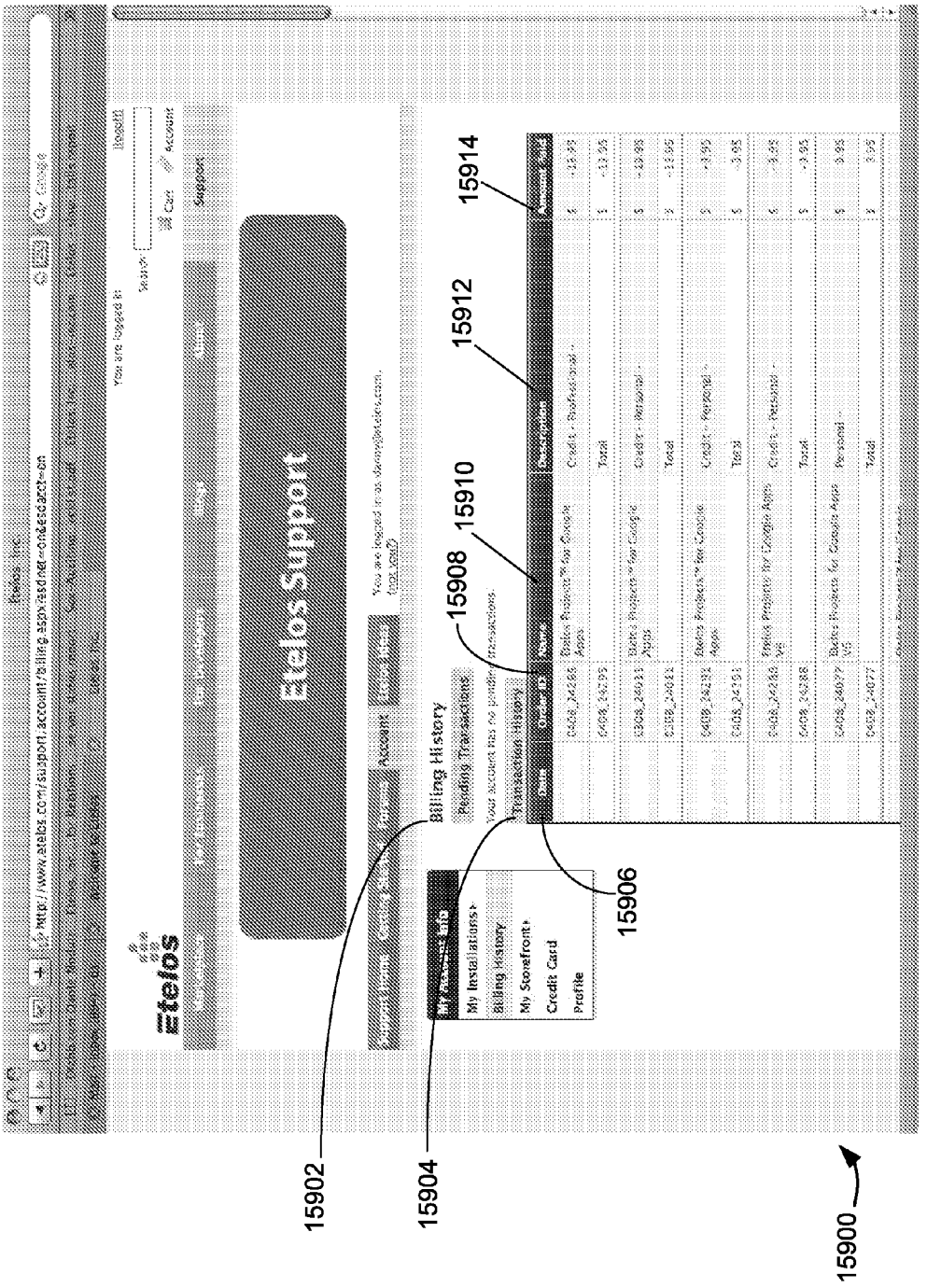

15900
15902
15904
15906
15908
15910
15912
15914
Etelos Support
Billing History
Pending Transactions
Transaction History
My Installations
Billing History
My Storefront
Credit Card
Profile

Figure 159

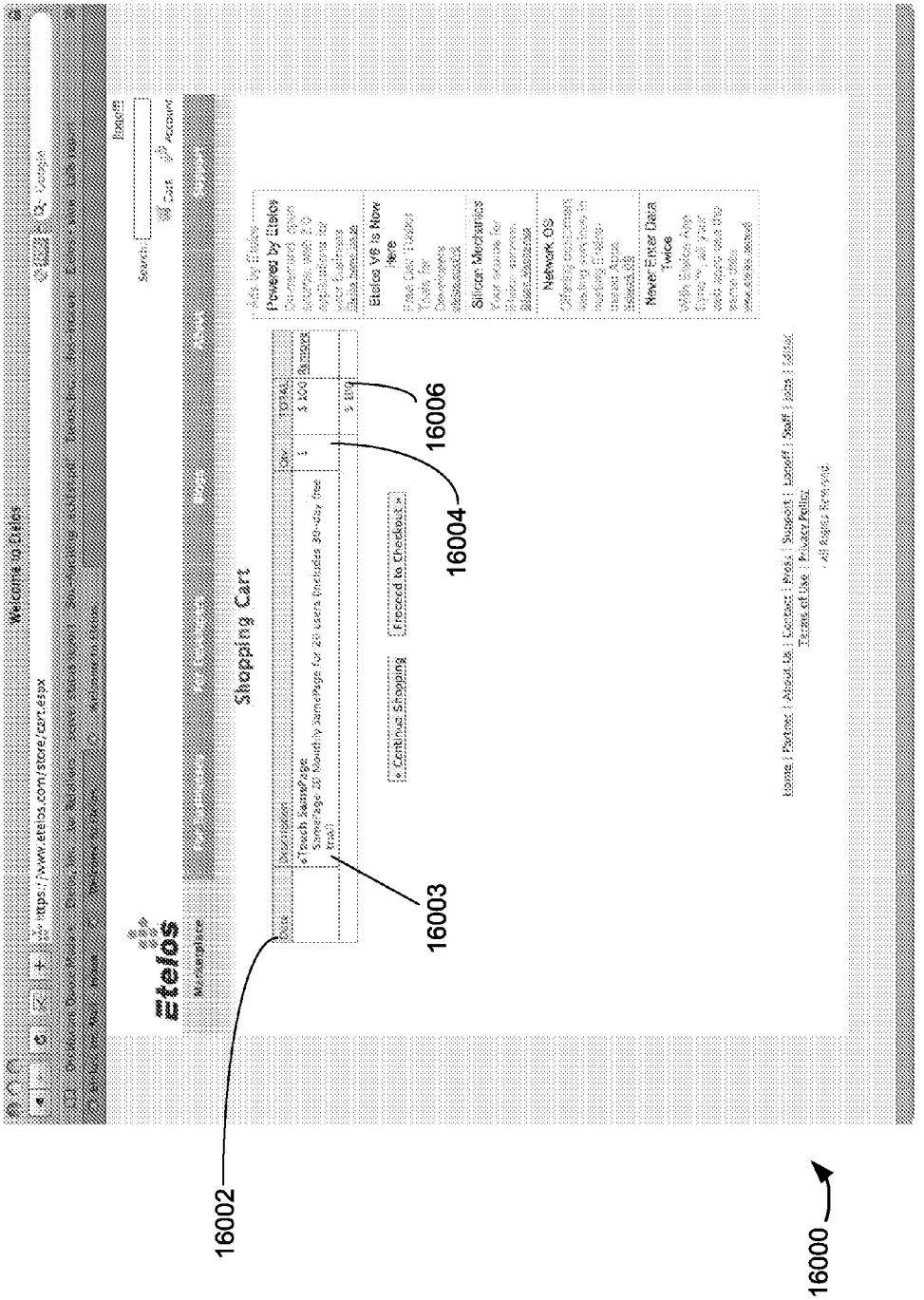


Figure 160

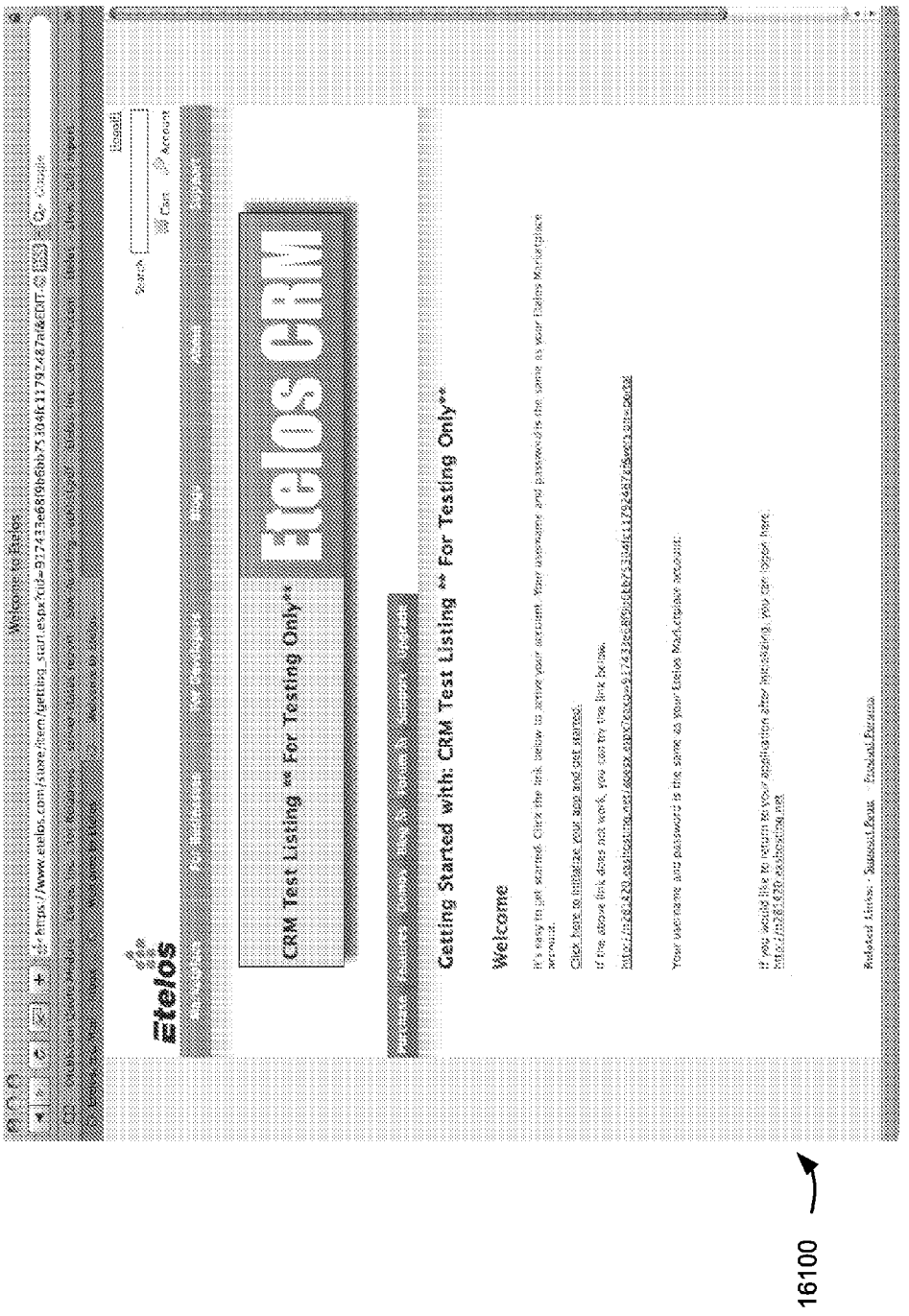

16100
Welcome to Etelos
Etelos
CRM Test Listing ** For Testing Only**
Etelos CRM
Getting Started with: CRM Test Listing ** For Testing Only**
Welcome

Figure 161

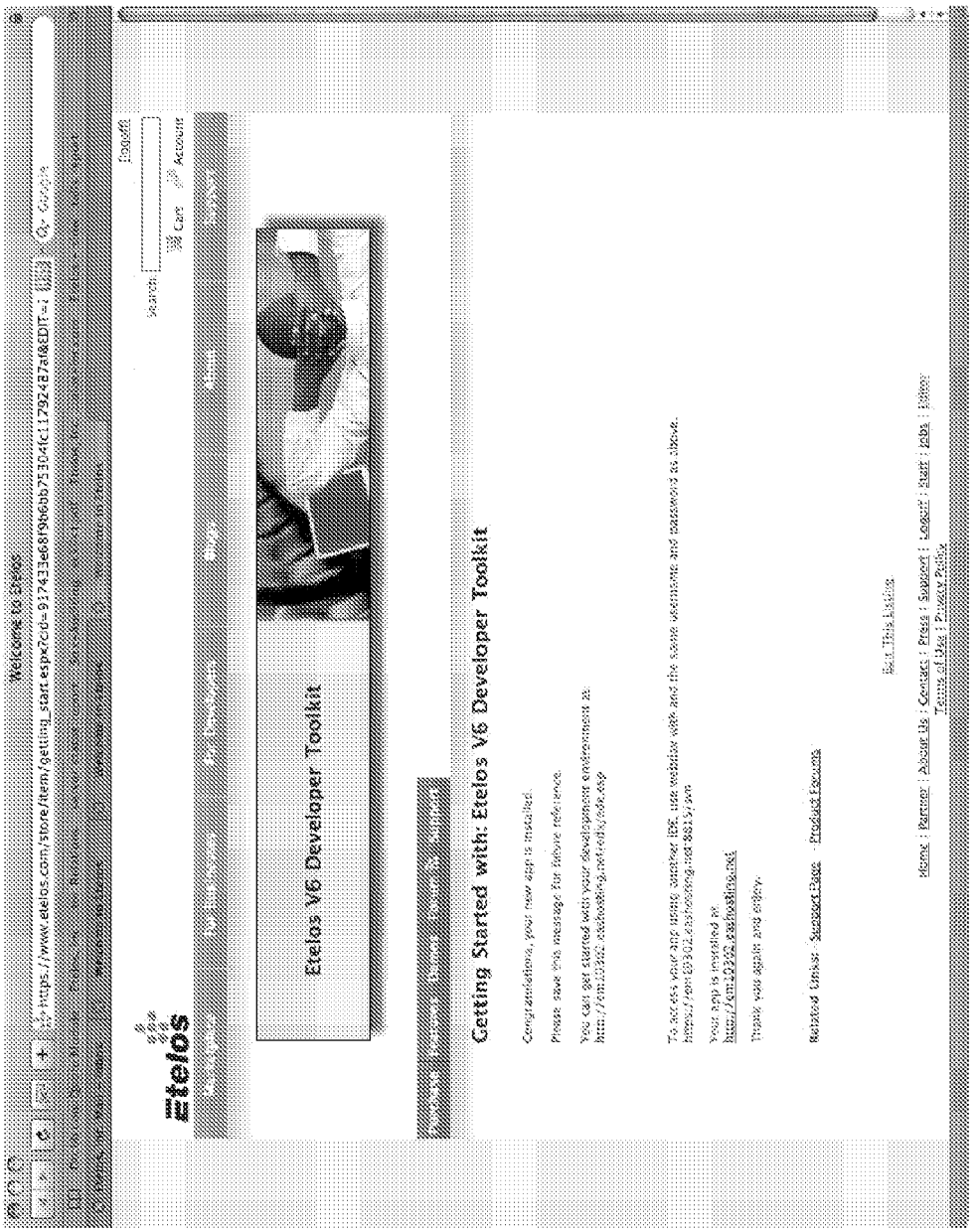

16200
Welcome to Etelos
Etelos
Etelos V6 Developer Toolkit
Getting Started with: Etelos V6 Developer Toolkit
Congratulations, your new app is installed.
Please save this message for future reference.
You can get started with your development environment at:
To access your app using another IDE, use webdav with and the same username and password as above.
Your app is installed at:
There was again and enjoy.
Related Links: Support Page - Product Forums
Back This Listing
Home | Partner | About Us | Contact | Press | Support | Logout | Start | Jobs | Terms of Use | Privacy Policy

Figure 162

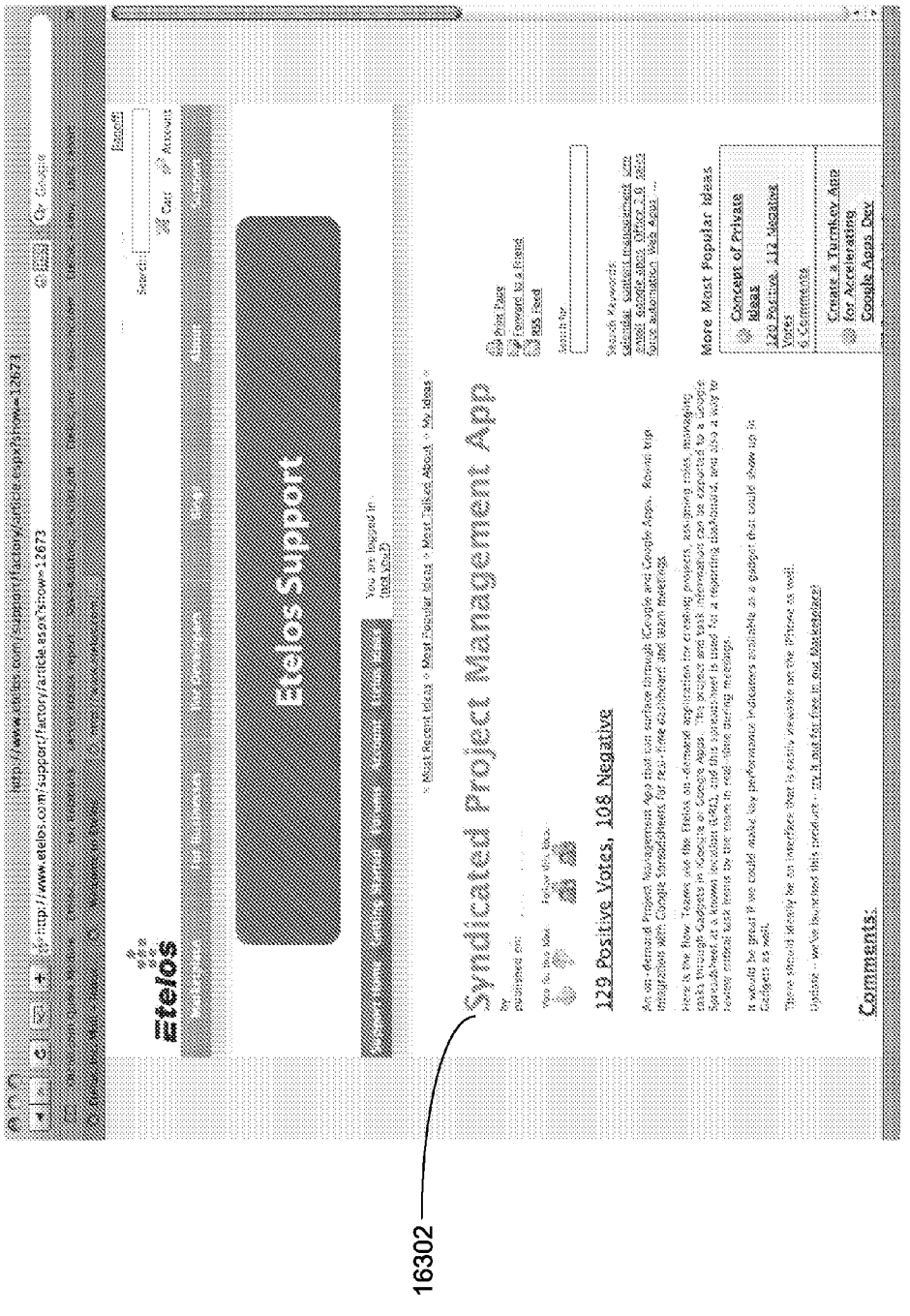

Etelos Support
Syndicated Project Management App
129 Positive Votes, 108 Negative
Comments:
16302
16300

Figure 163

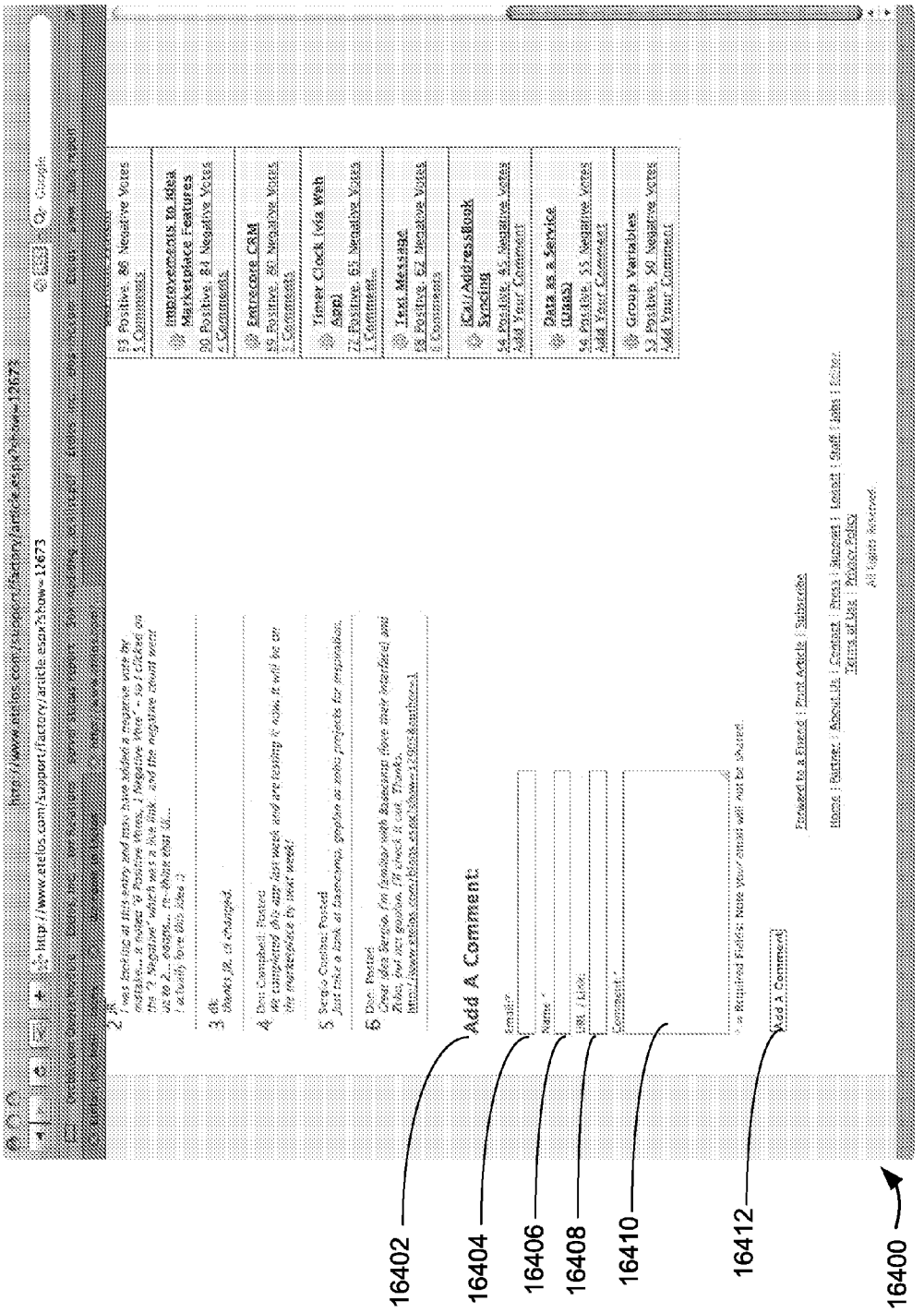


Figure 164

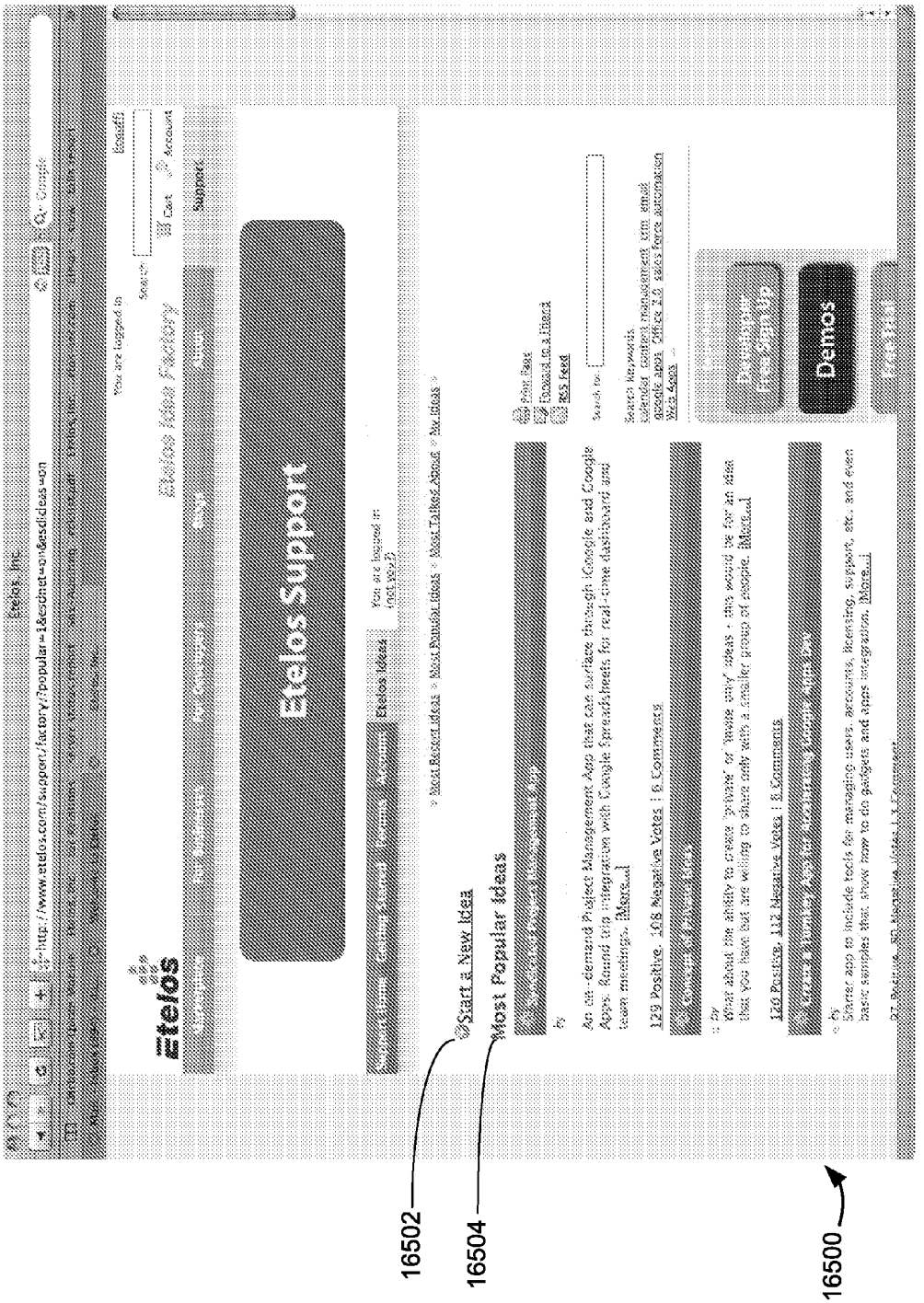

Etelos, Inc.
Etelos
Etelos Support
Start a New Idea
Most Popular Ideas
16502
16504
16500

Figure 165

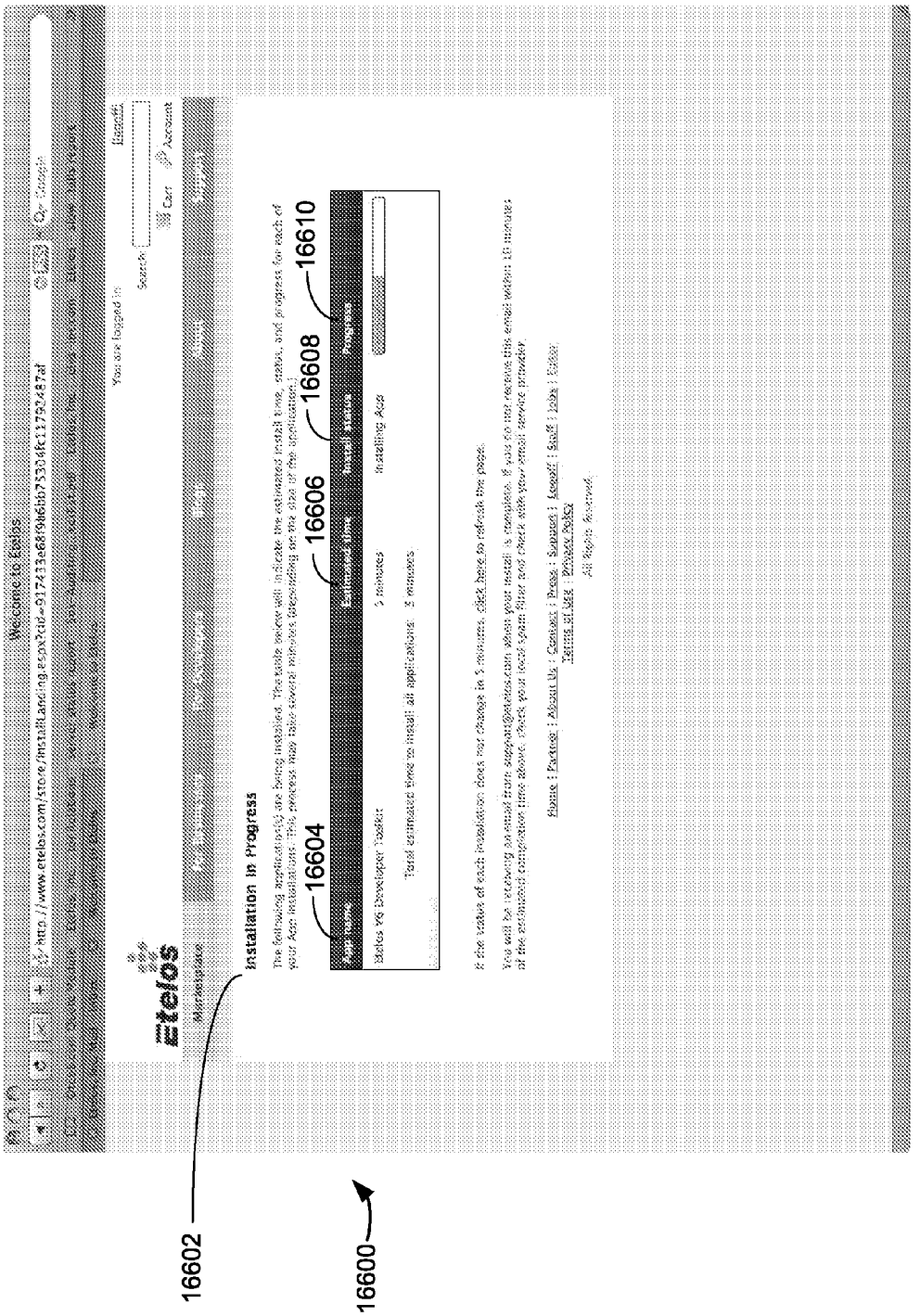

Welcome to Etelos
Etelos
Marketplace
Installation in Progress
16602
16600
16604
16606
16608
16610

Figure 166

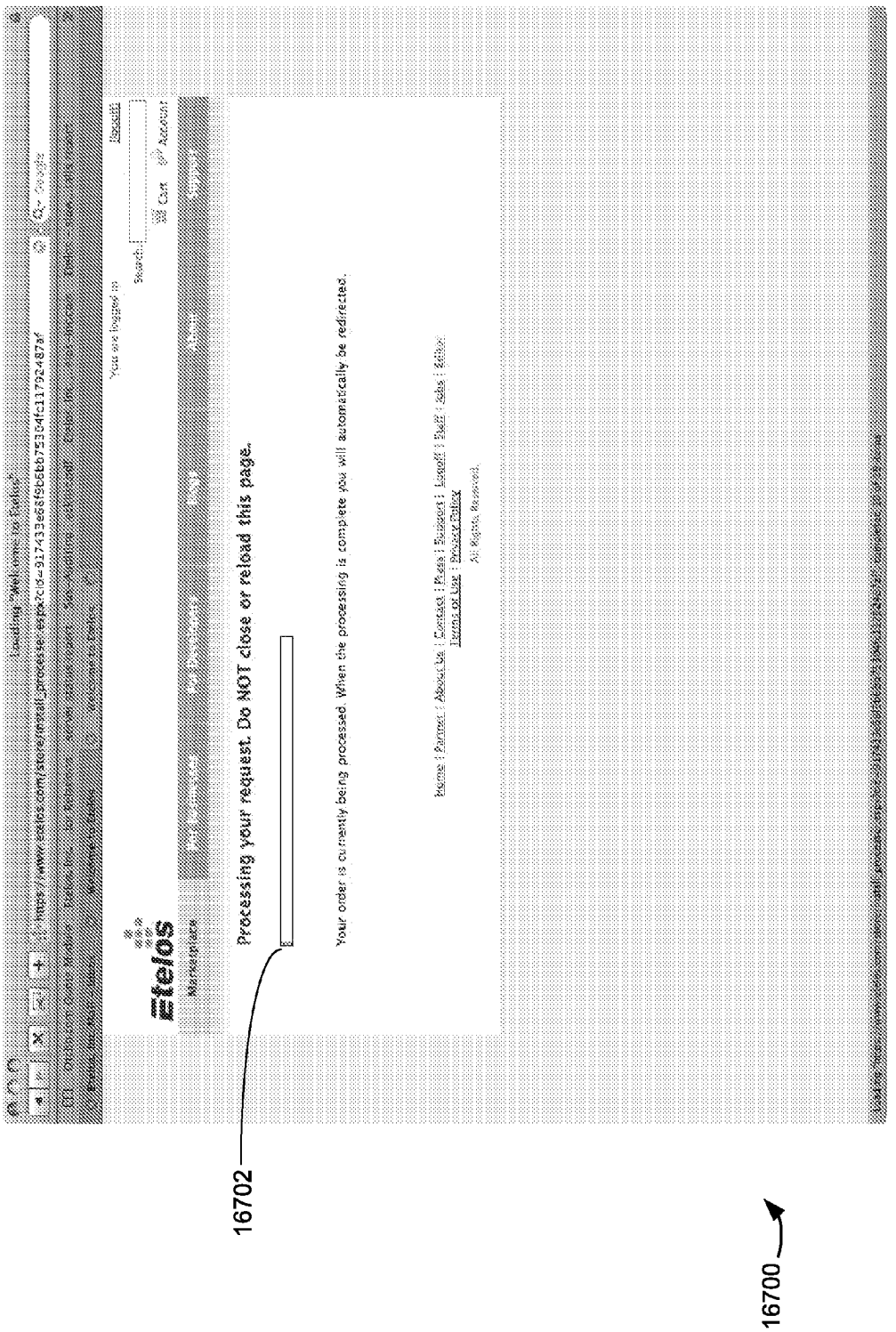
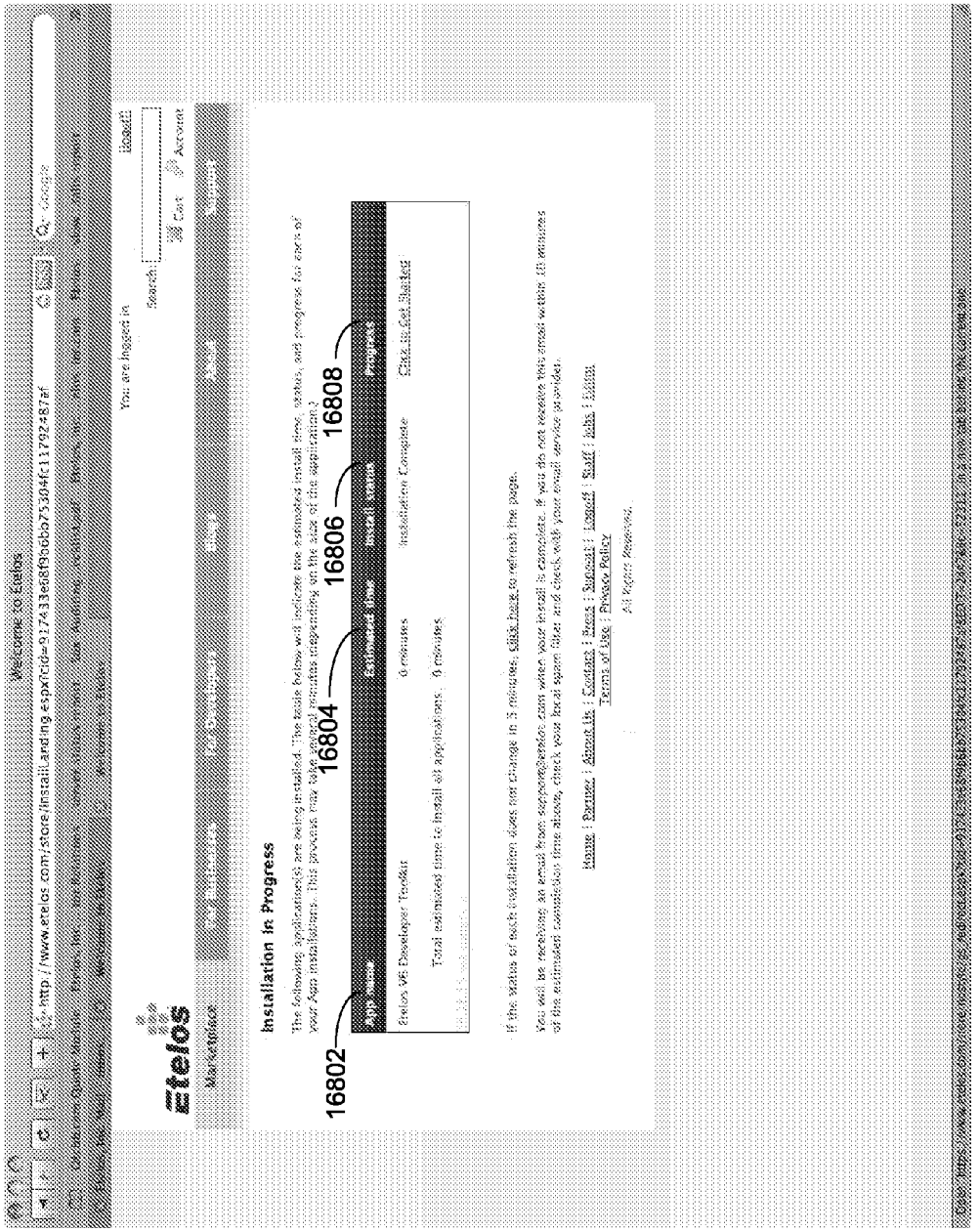

Processing your request. Do NOT close or reload this page.
Your order is currently being processed. When the processing is complete you will automatically be redirected.
16702
16700

Figure 167



16800

Figure 168

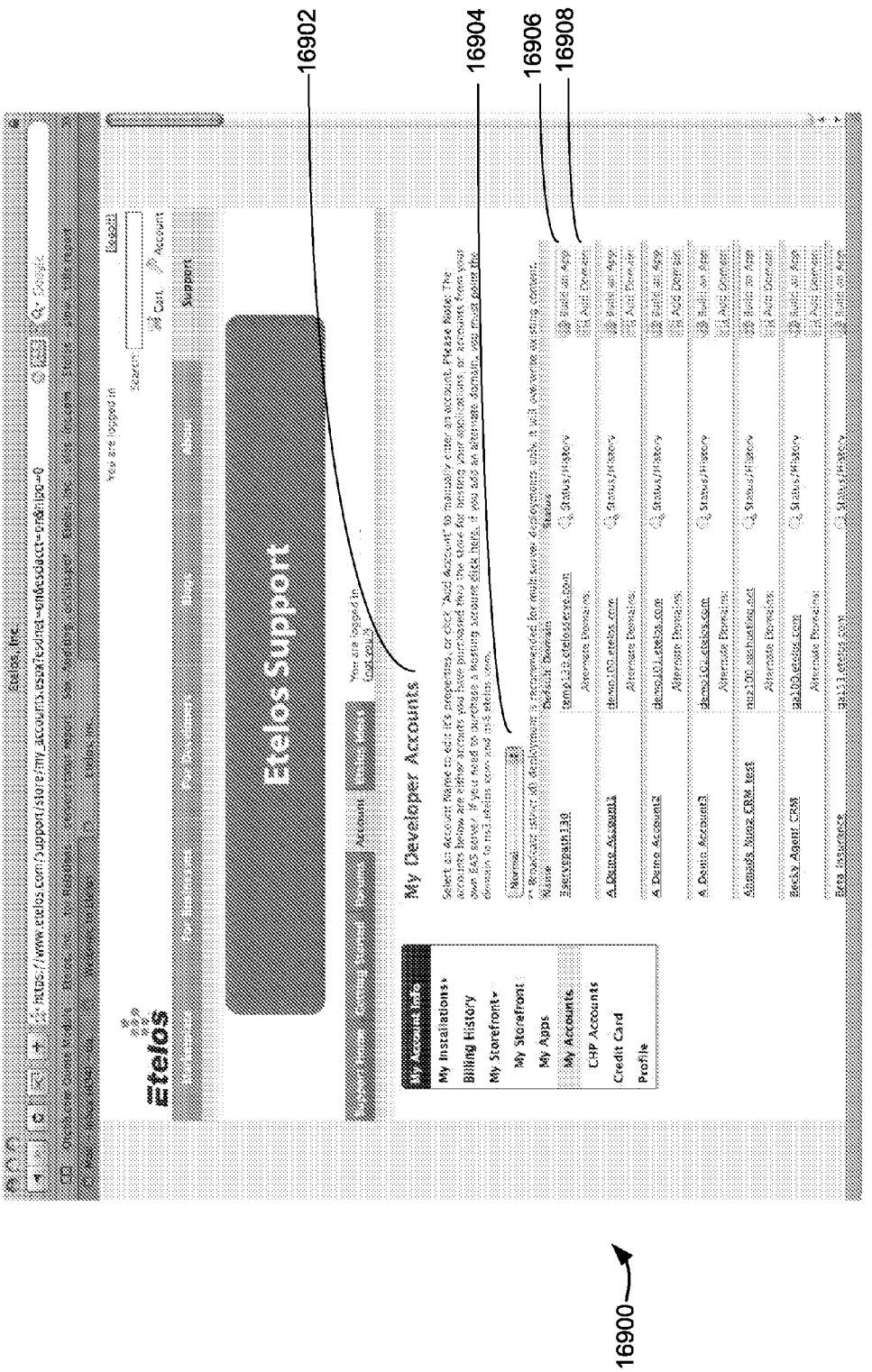

16902
16904
16906
16908
16900
Etelos Support
My Developer Accounts

Figure 169

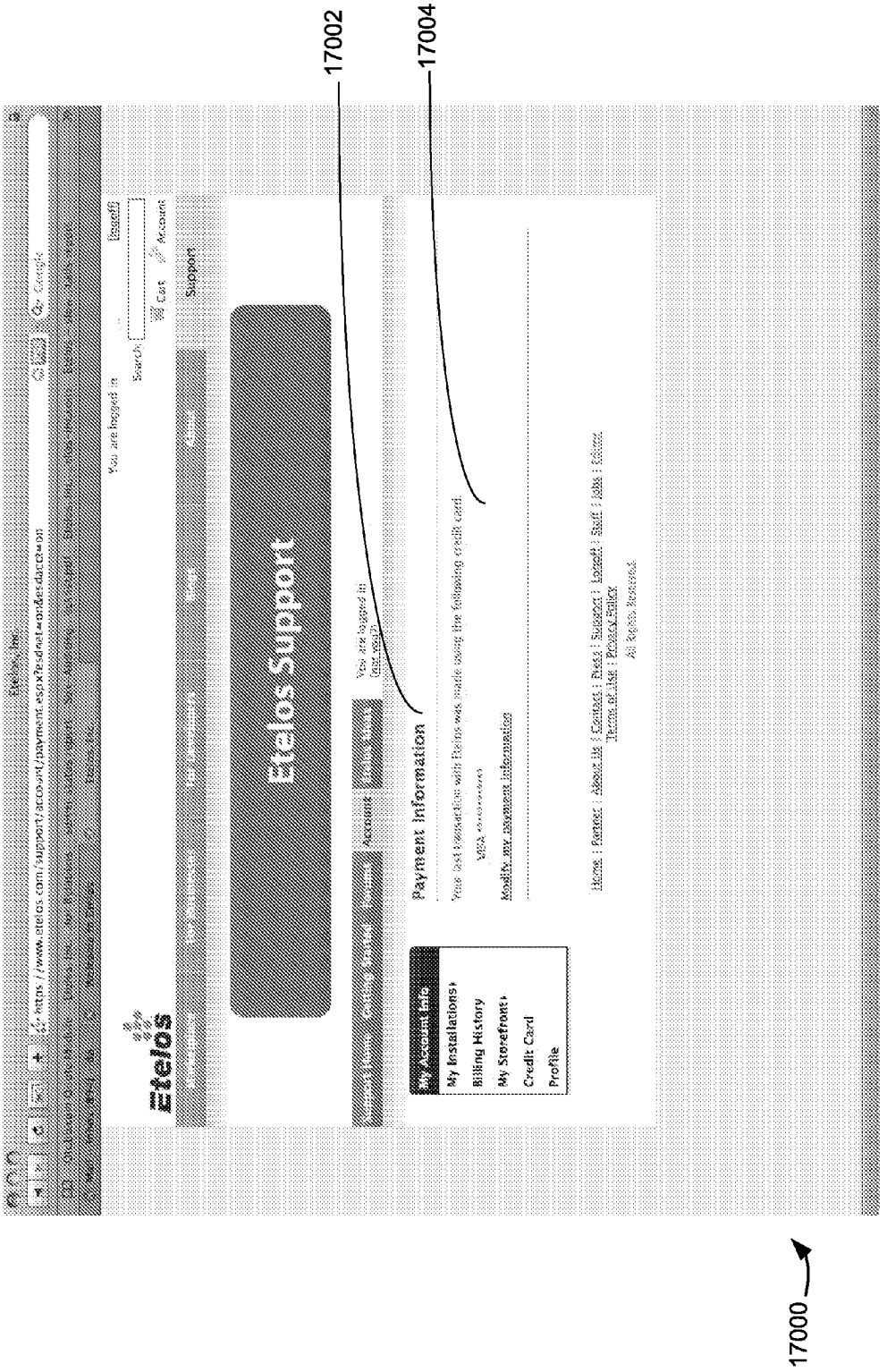

17002
17004
17000
Etelos Support
Payment Information
My Account Info
My Installations
Billing History
My Storefronts
Credit Card
Profile

Figure 170

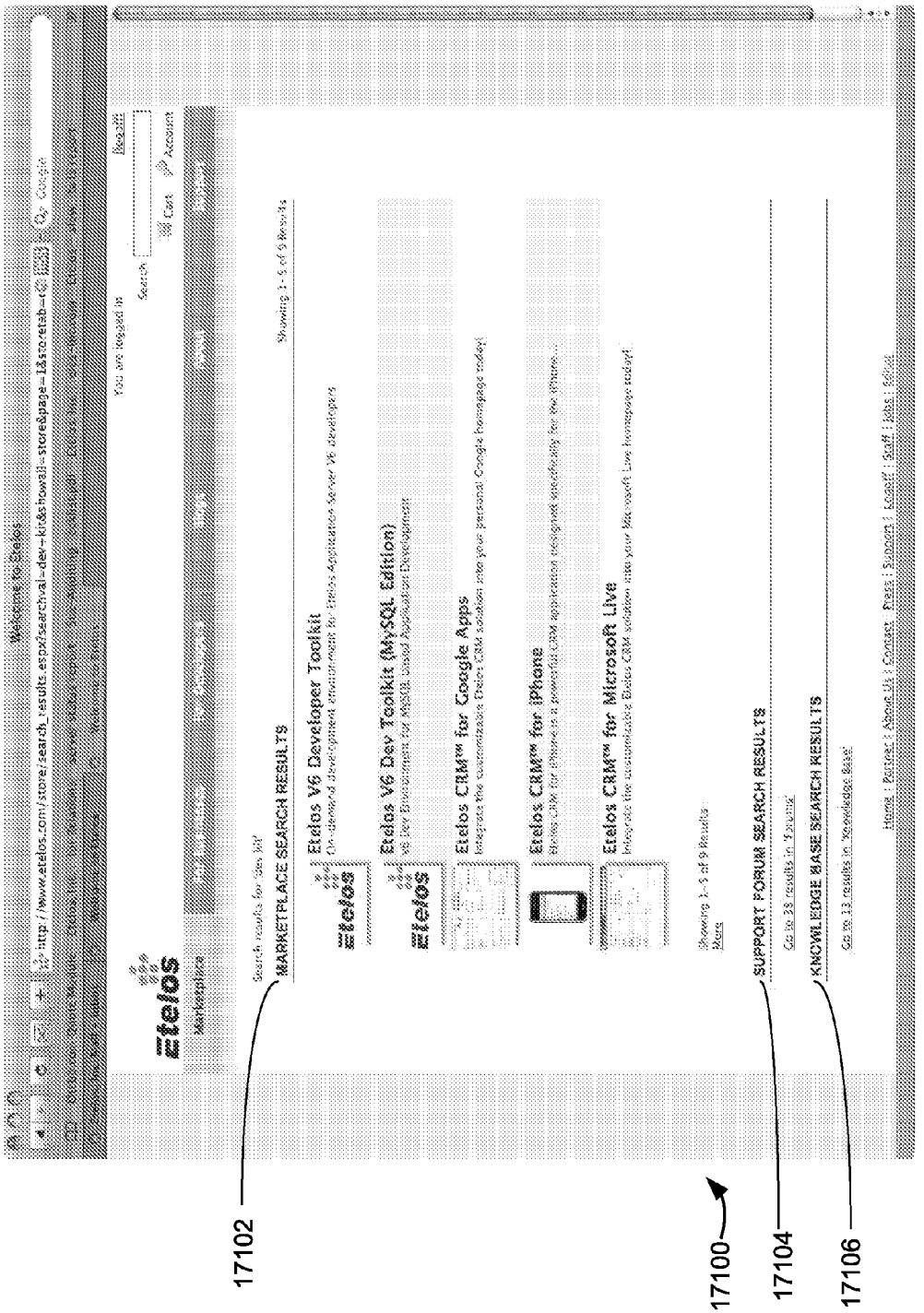

17102
17100
17104
17106
MARKETPLACE SEARCH RESULTS
Etelos V6 Developer Toolkit
Etelos V6 Dev Toolkit (MySQL Edition)
Etelos CRM™ for Google Apps
Etelos CRM™ for iPhone
Etelos CRM™ for Microsoft Live
SUPPORT FORUM SEARCH RESULTS
KNOWLEDGE BASE SEARCH RESULTS

Figure 171

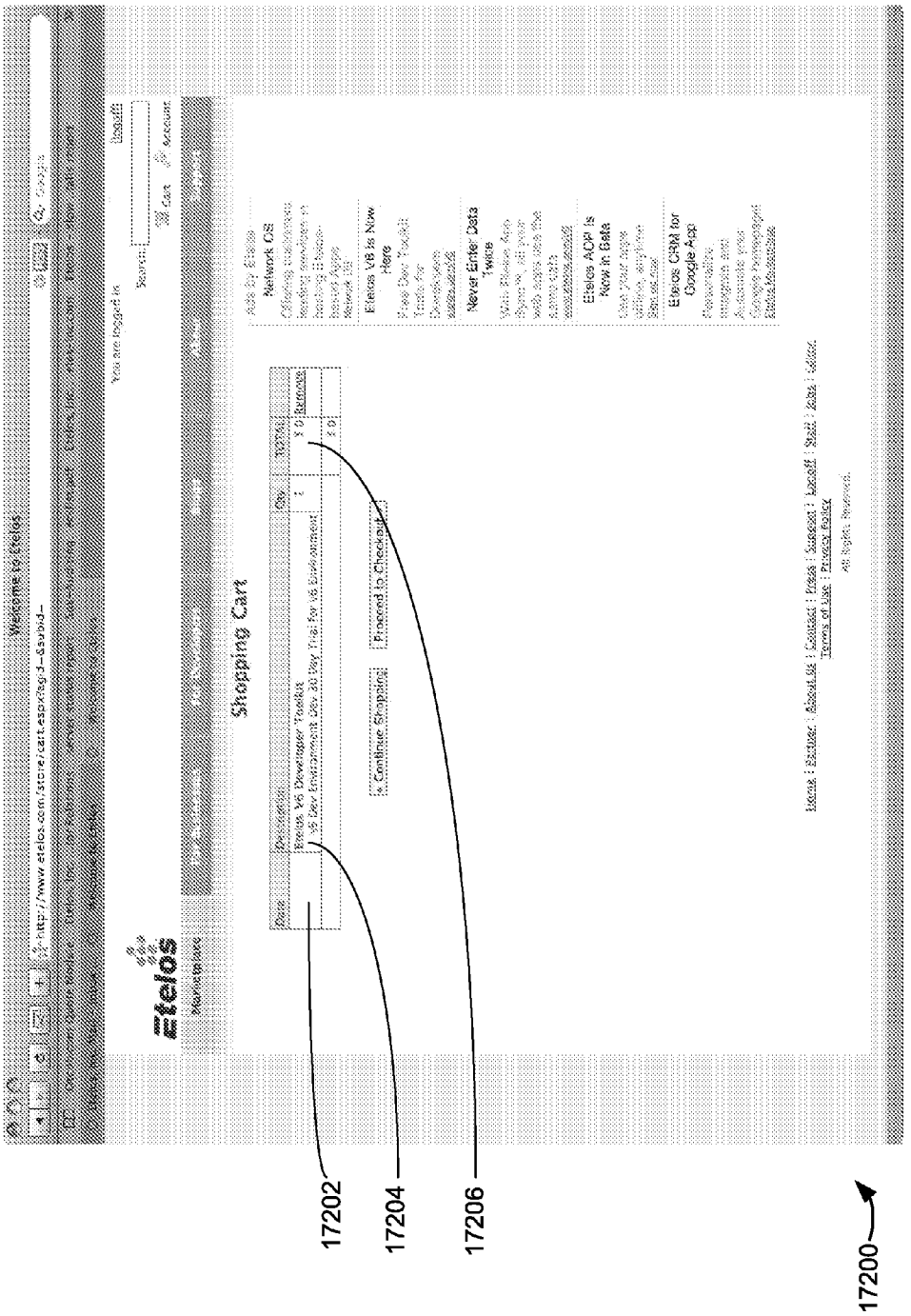


Figure 172

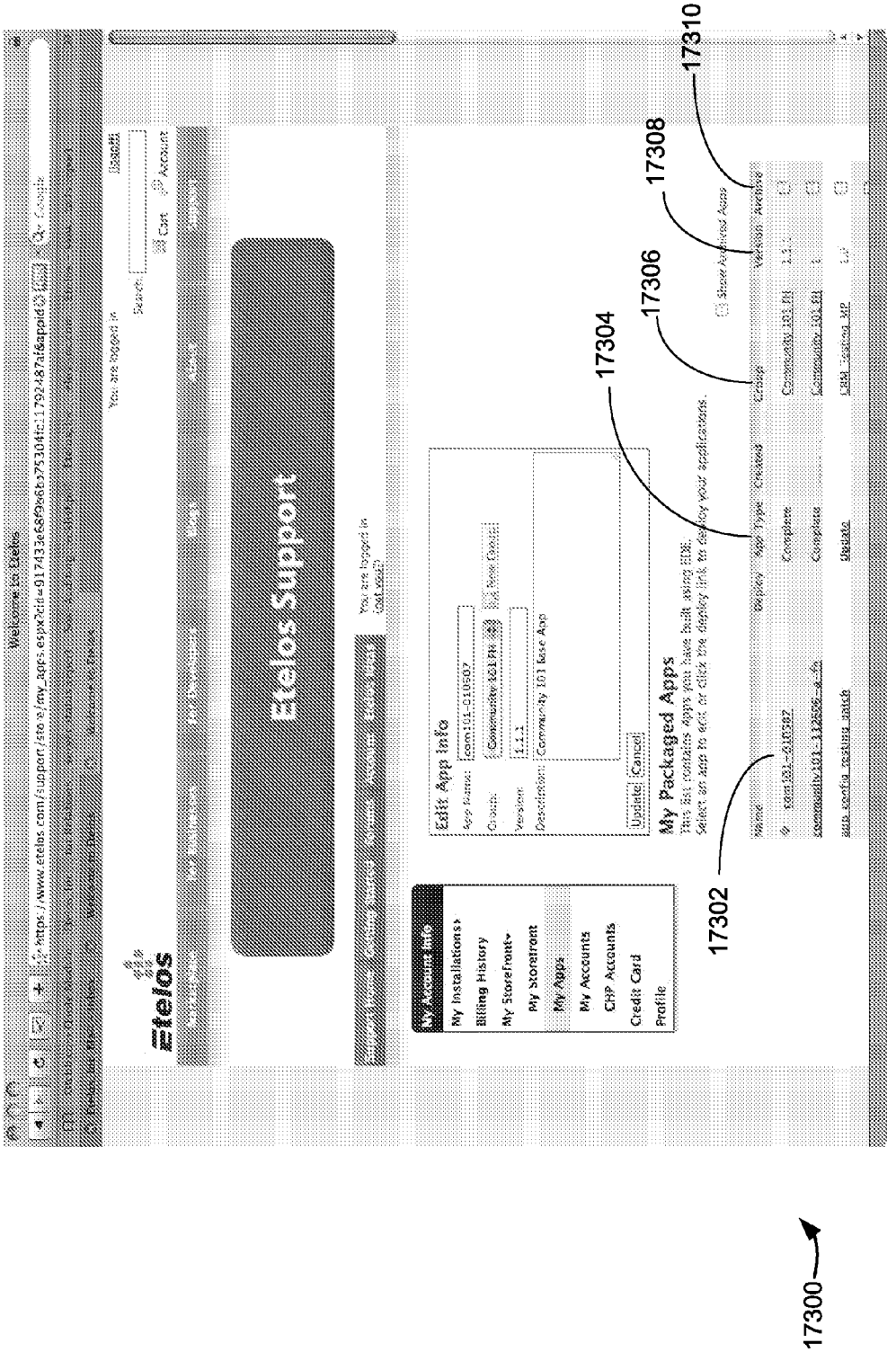

17300
17302
17304
17306
17308
17310
Welcome to Etelos
Etelos Support
My Account Info
My Installations
Billing History
My StoreFronts
My Storefront
My Apps
My Accounts
CRP Accounts
Credit Card
Profile
Edit App Info
My Packaged Apps

Figure 173

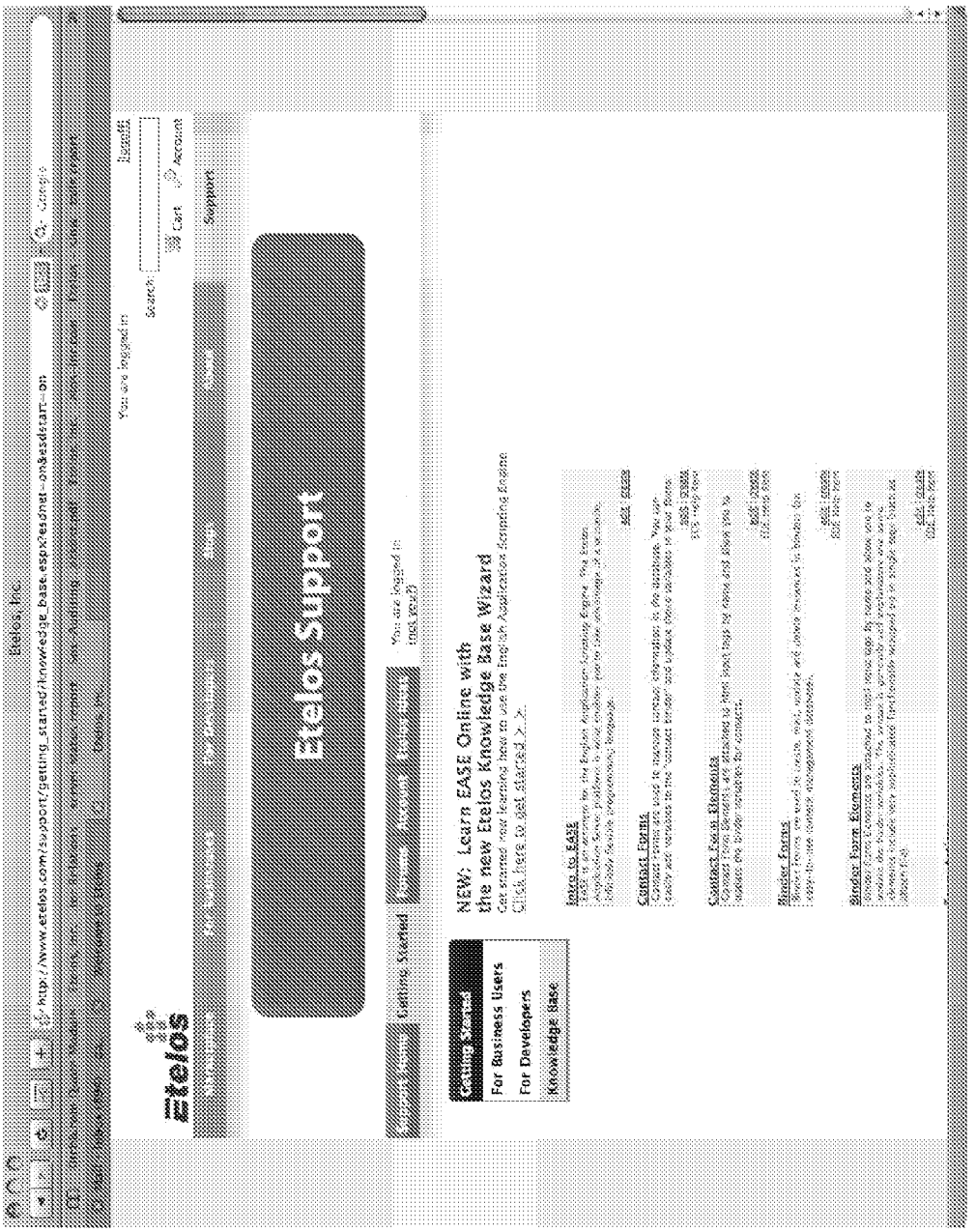

Etelos, Inc.
Etelos Support
Getting Started
For Business Users
For Developers
Knowledge Base
NEW: Learn EASE Online with
the new Etelos Knowledge Base Wizard
Click here to get started > >
17400

Figure 174

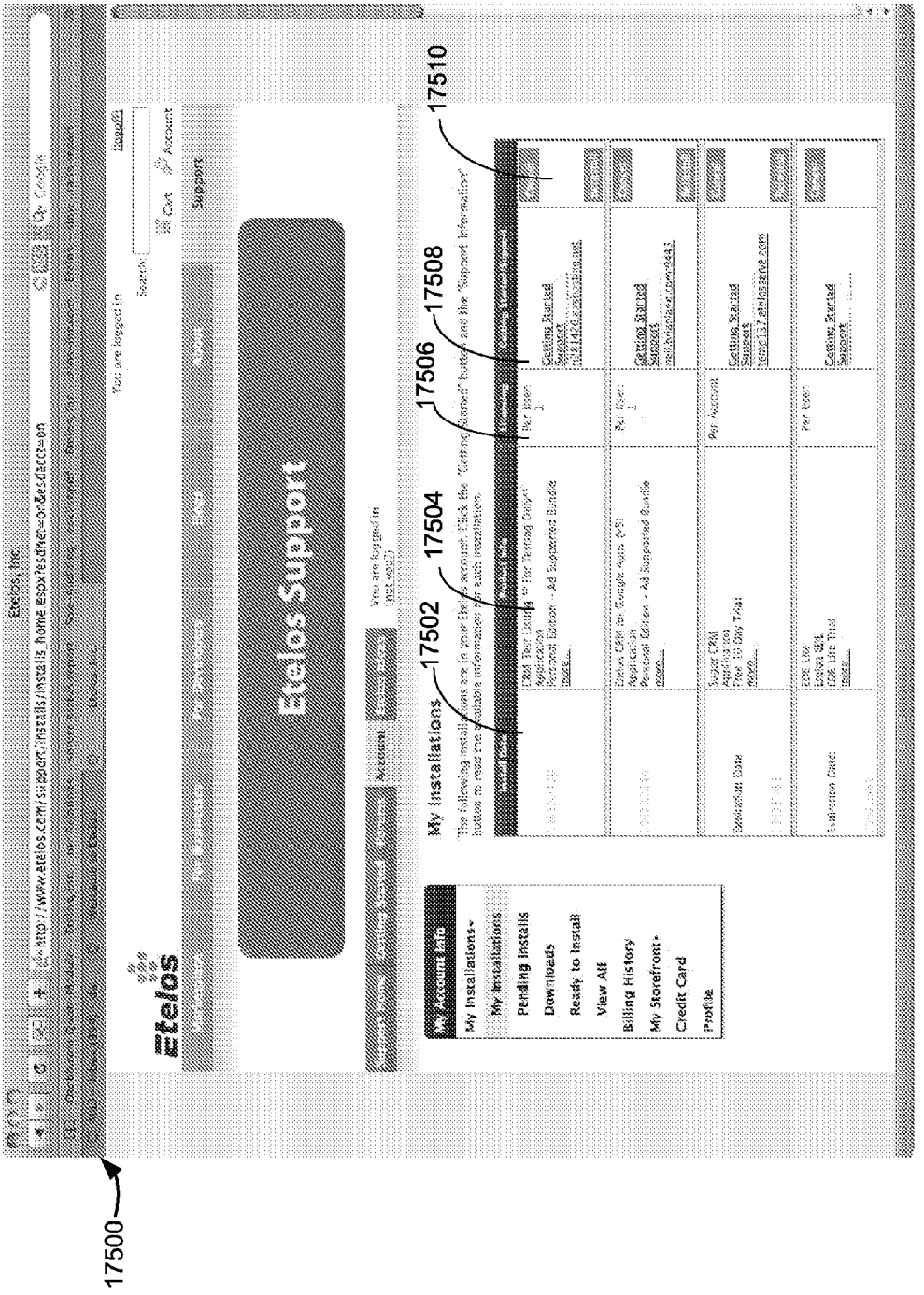

17500
17502
17504
17506
17508
17510
Etelos Support
My Installations
My Account Info
My Installations
Pending Installs
Downloads
Ready to Install
View All
Billing History
My Storefront
Credit Card
Profile

Figure 175

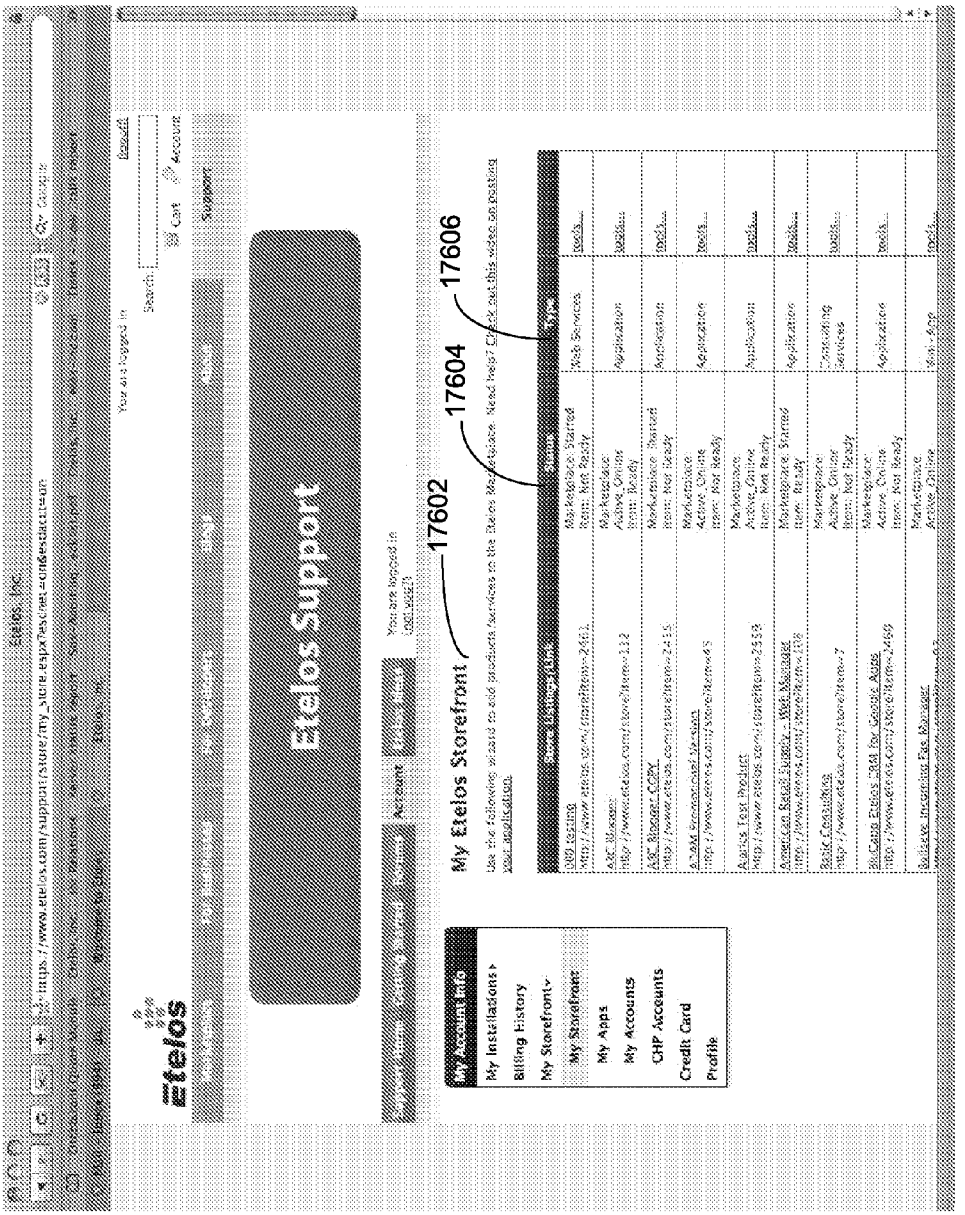

17600
17602
17604
17606
Etelos Support
My Etelos Storefront
My Account Info
My Installations
Billing History
My Storefront
My Apps
My Accounts
CHP Accounts
Credit Card
Profile

Figure 176

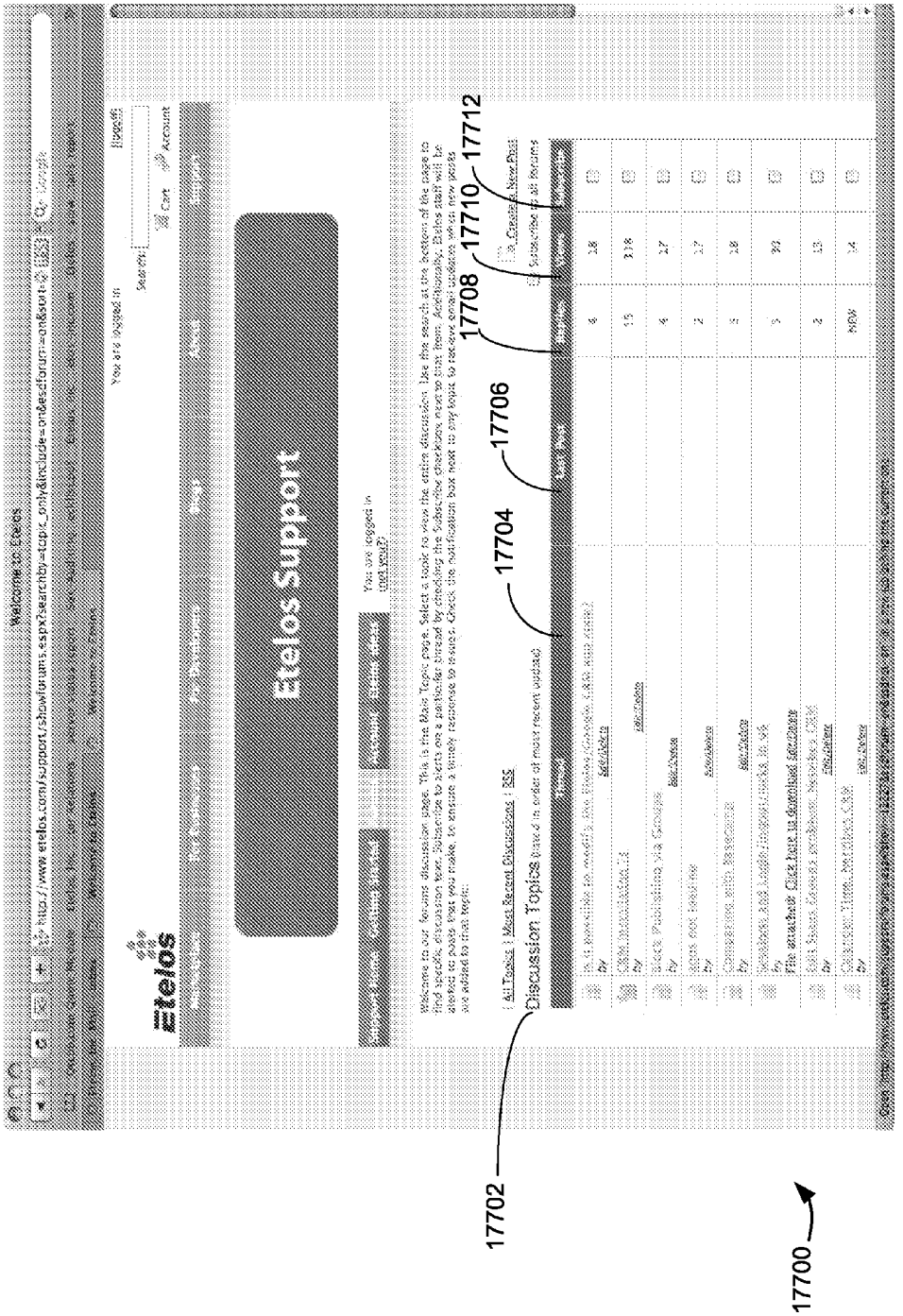

17700
17702
17704
17706
17708
17710
17712
Etelos Support
Discussion Topics

Figure 177

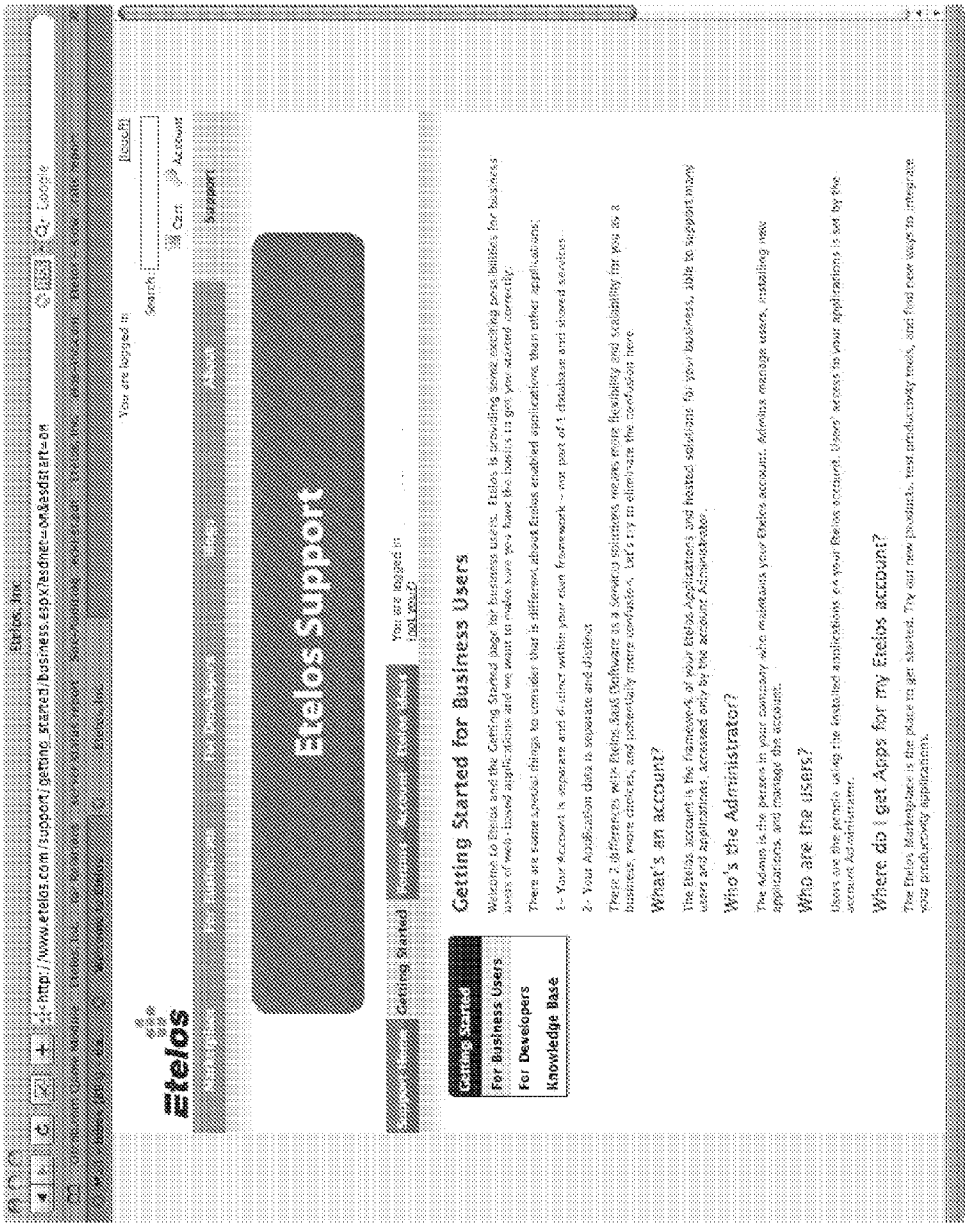

17800
Etelos, Inc.
http://www.etelos.com/support/getting_started/business.aspx?redirect=0&asdstart=on
Etelos
Support
You are logged in
Search
Cart
Account
Etelos Support
Getting Started
Getting Started for Business Users
Getting Started
For Business Users
For Developers
Knowledge Base
What's an account?
Who's the Administrator?
Who are the users?
Where do I get Apps for my Etelos account?

Figure 178

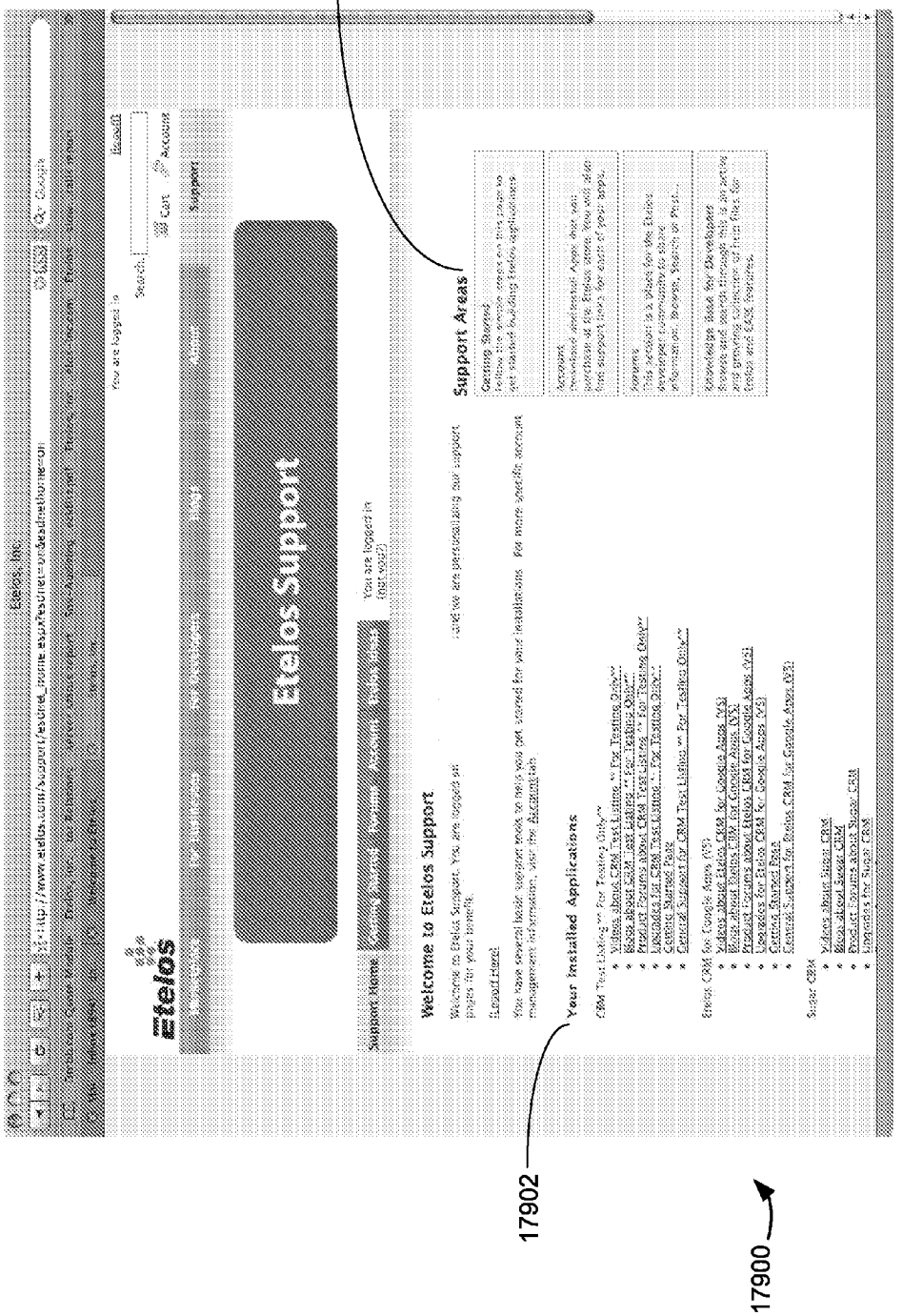

17904
17902
17900
Etelos Support
Welcome to Etelos Support
Your Installed Applications
Support Areas

Figure 179

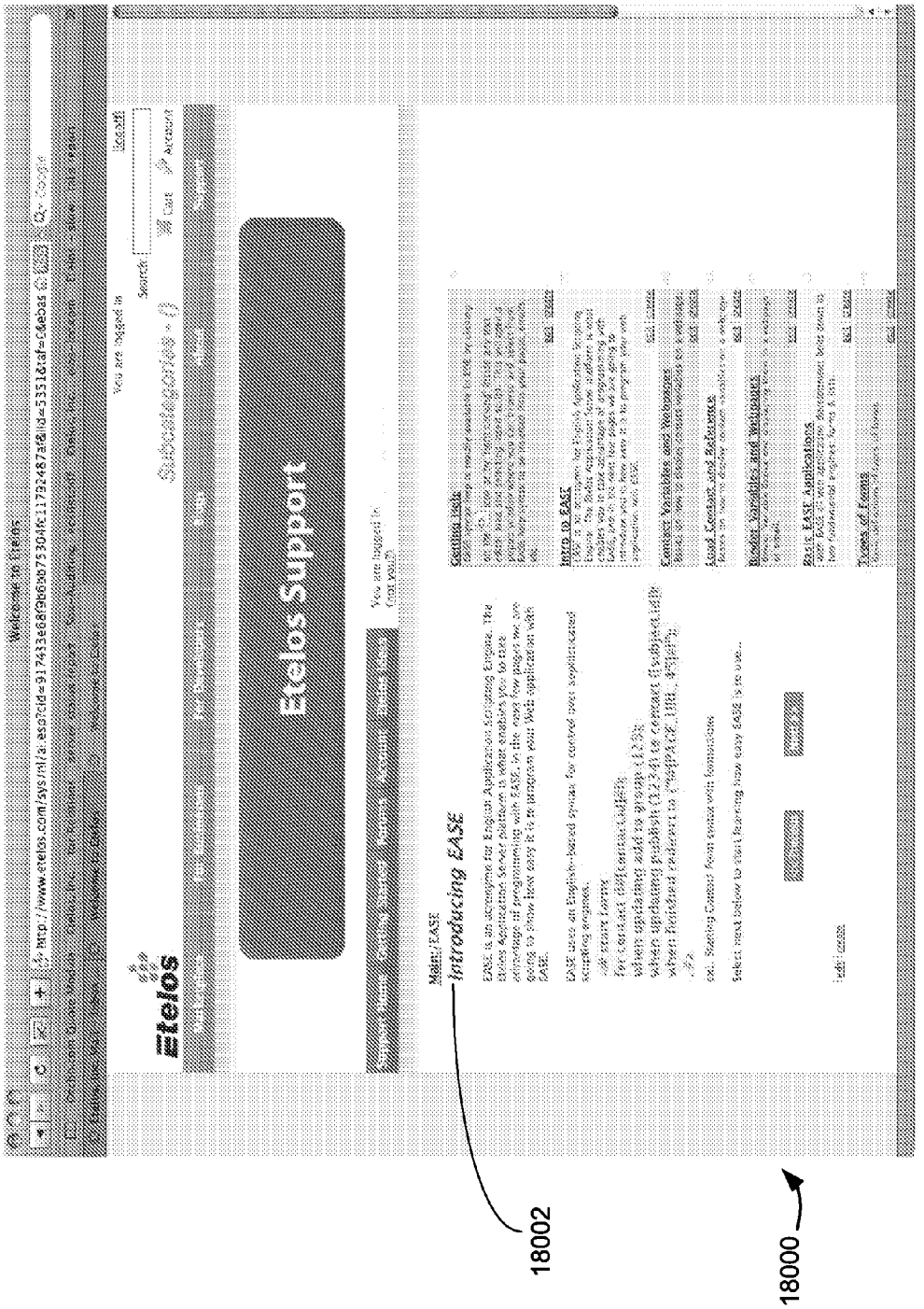

Welcome to Etelos
Etelos
Etelos Support
Main: EASE
Introducing EASE
18002
18000

Figure 180

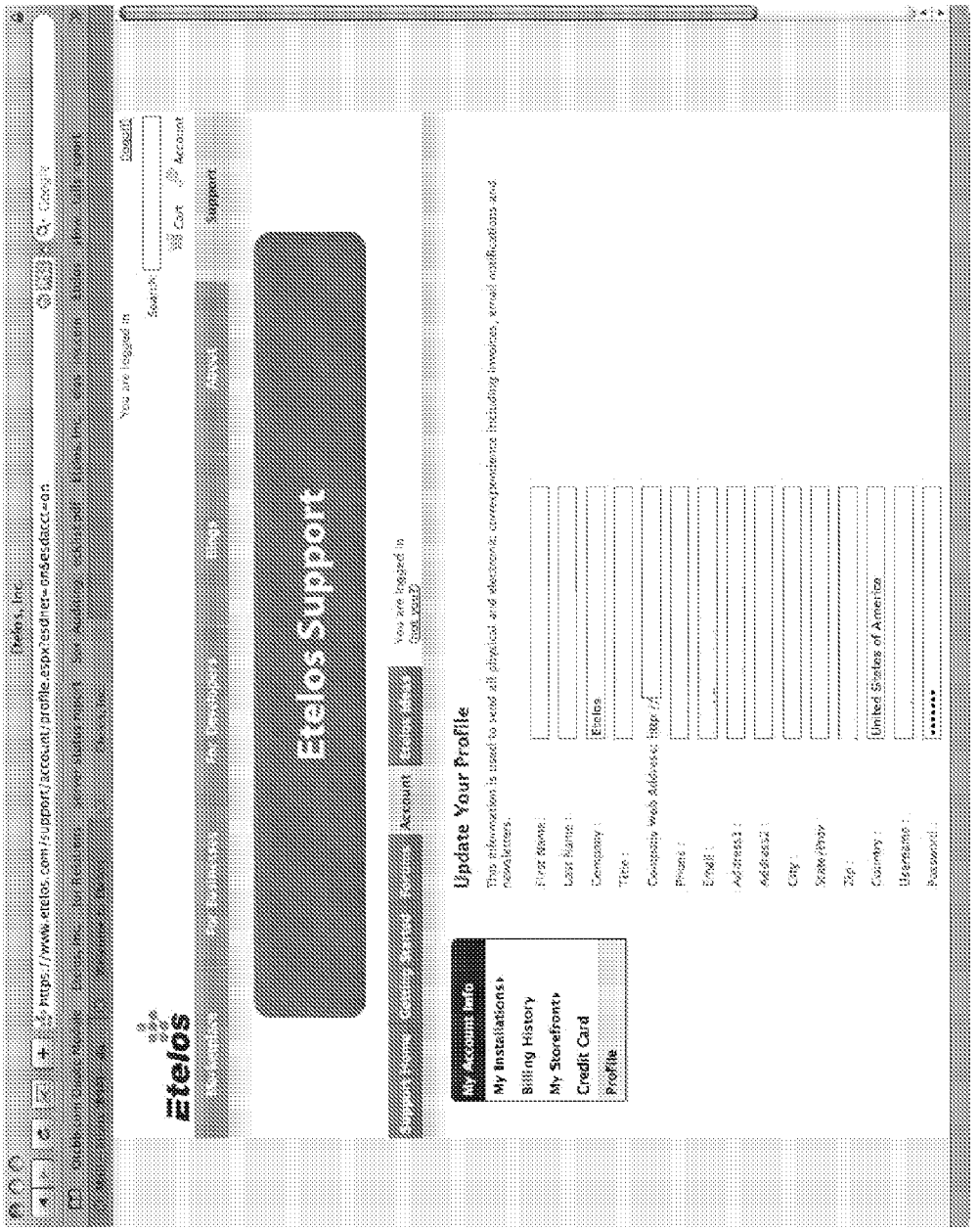


Figure 181

18100

BT Eklas Storefront : Eklas Marketplace
http://www.eklas.com/marketplace/storefront/?mid=7&storetab=on
BT
Web21C SDK
Marketplace
Community
Apps
About Us
Search
Sign up for Eklas AOP Beta today
Use your Web apps
Offline, Online, Anytime
Featured Apps
Search
Categories
Eklas Hosting and Dev Environment for BT Web21C SDK
Eklas CRM for Google App
Never Enter Data Twice
18200

Figure 182

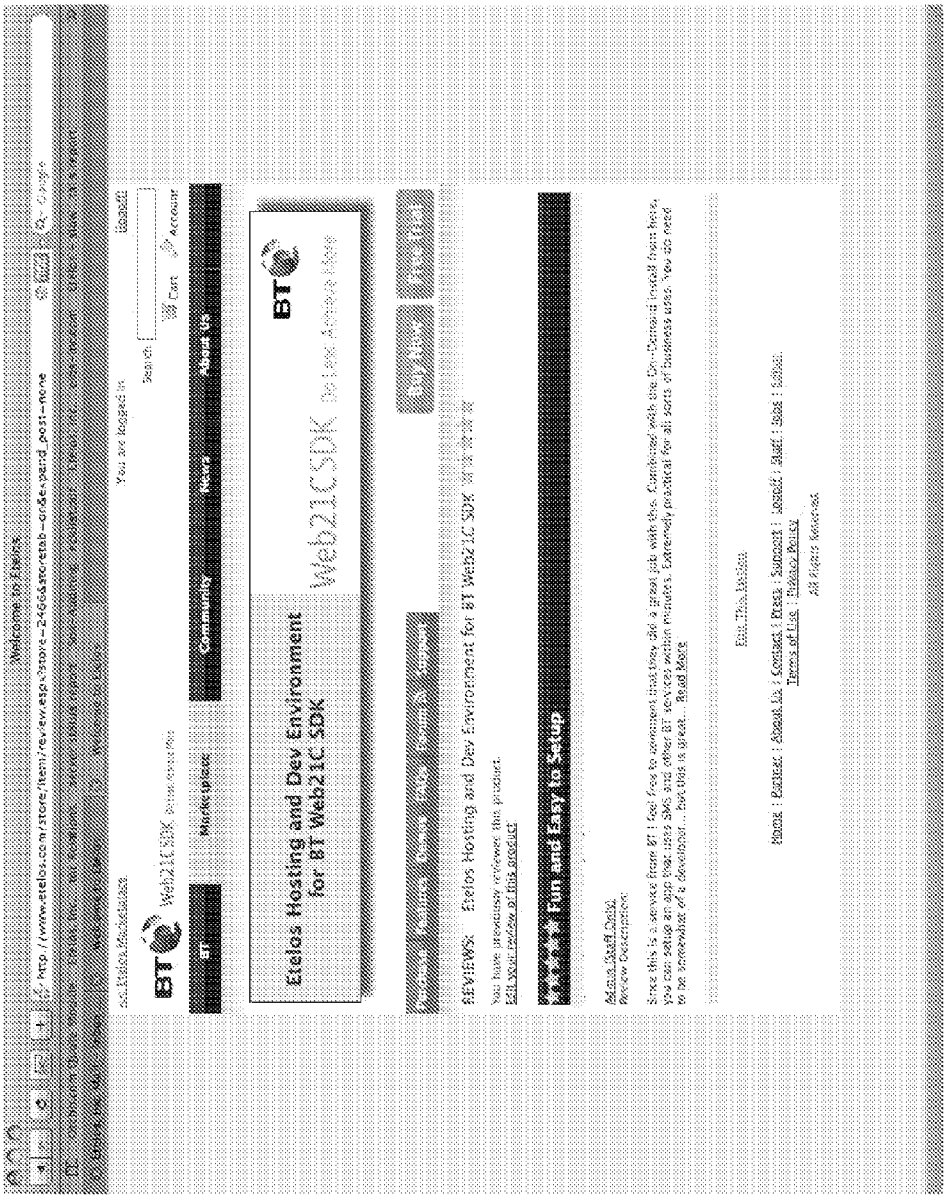

18300
Welcome to Etelos
BT
Web21C SDK
Marketplace
Community
Etelos Hosting and Dev Environment for BT Web21C SDK
Web21C SDK
Run and Easy to Setup

Figure 183

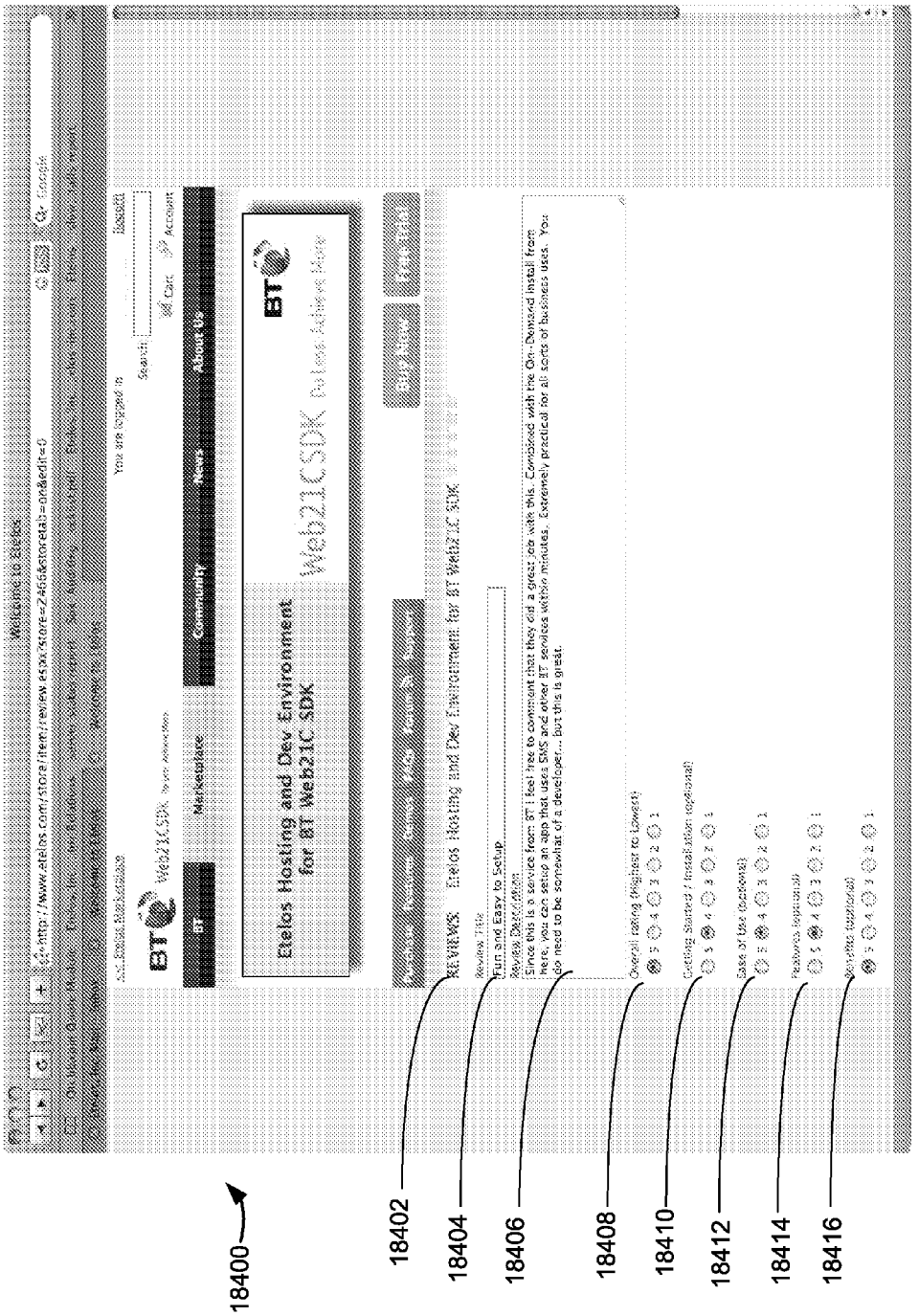

18400
18402
18404
18406
18408
18410
18412
18414
18416
Welcome to Etelos
BT
Web21C SDK
Marketplace
Community
News
About Us
Etelos Hosting and Dev Environment for BT Web21C SDK
Web21C SDK
REVIEWS
Etelos Hosting and Dev Environment for BT Web21C SDK
Fun and Easy to Setup
Overall rating (highest to lowest)
Getting Started / Installation (optional)
Ease of Use (optional)
Features (optional)
Reliability (optional)

Figure 184

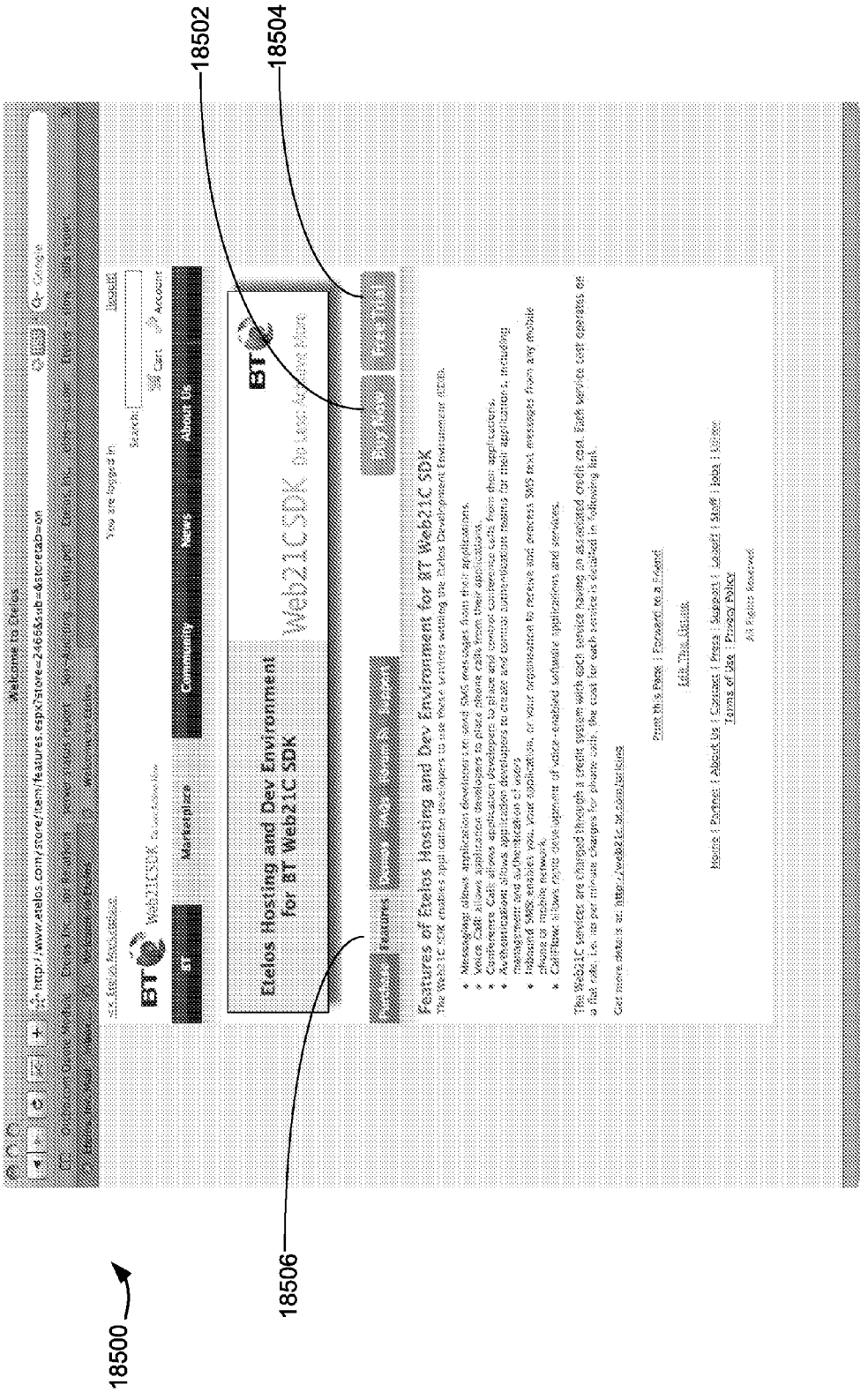

18500
18502
18504
18506
Etelos Hosting and Dev Environment for BT Web21C SDK
Web21CSDK
Features of Etelos Hosting and Dev Environment for BT Web21C SDK

Figure 185

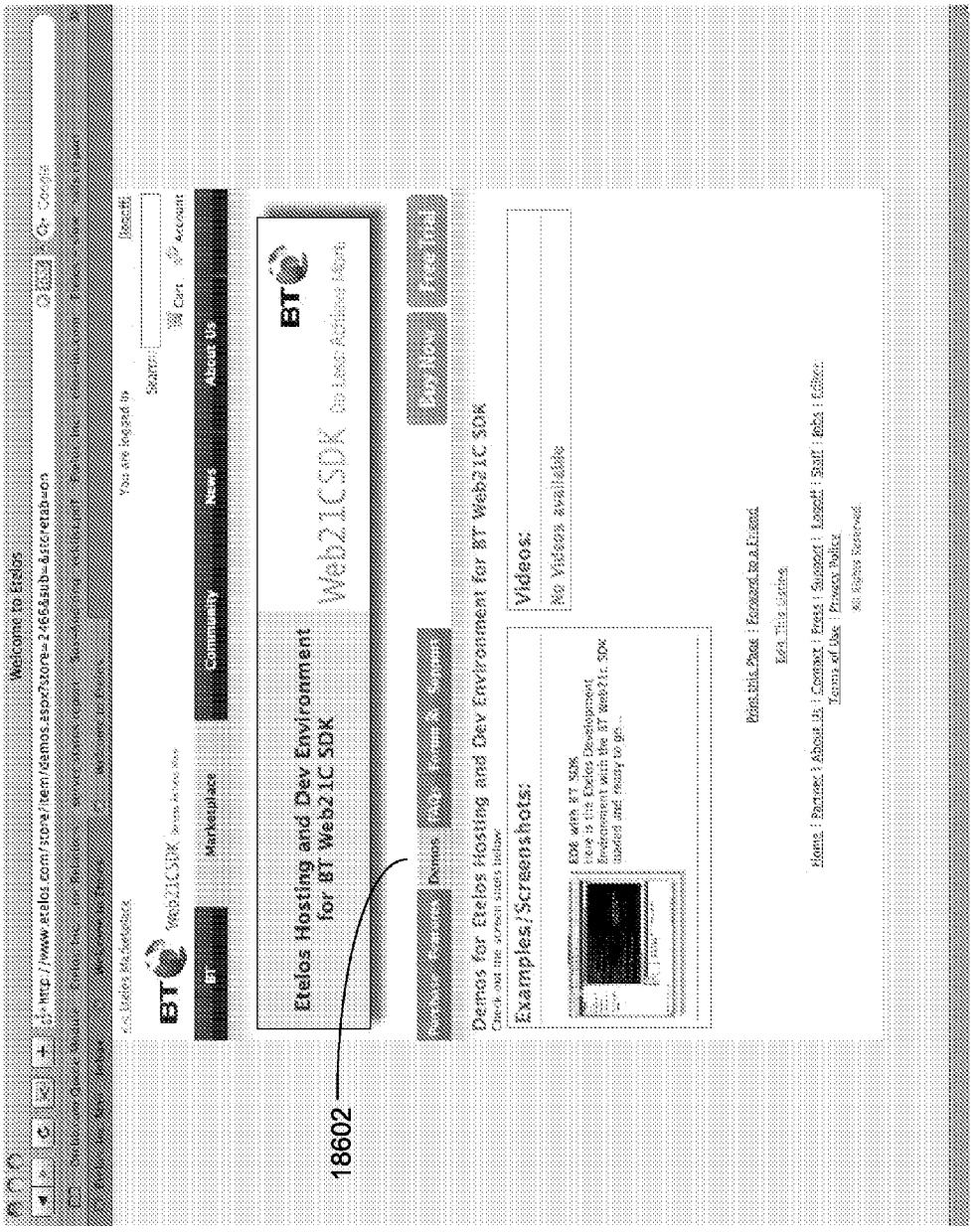

18600
18602
Welcome to Etelos
Etelos Hosting and Dev Environment for BT Web2IC SDK
Demos for Etelos Hosting and Dev Environment for BT Web2IC SDK
Examples/Screenshots:
Videos:
No Videos available
Community
News
About Us
Contact
Feedback

Figure 186

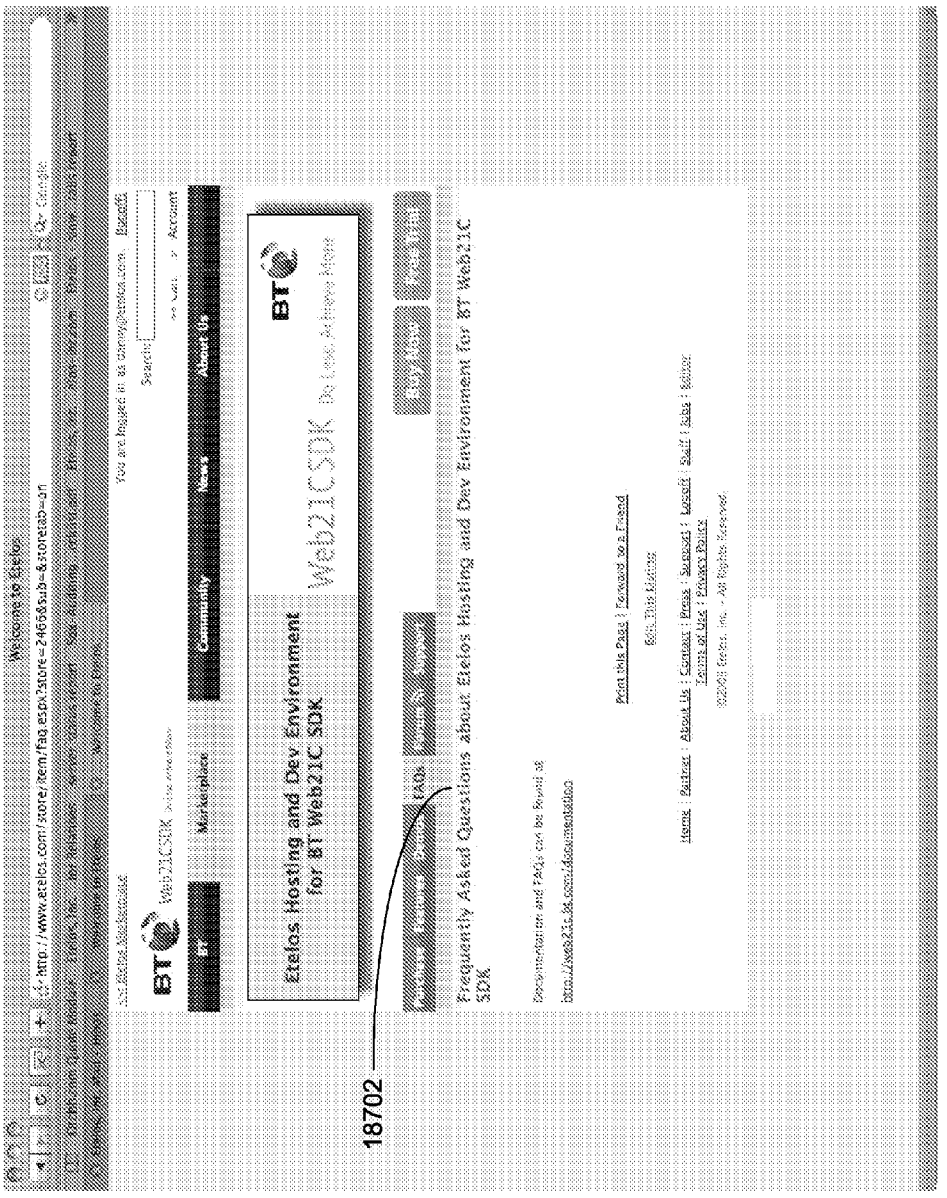

18702
18700
Etelos Hosting and Dev Environment for BT Web21C SDK
Web21CSDK
Frequently Asked Questions about Etelos Hosting and Dev Environment for BT Web21C SDK

Figure 187

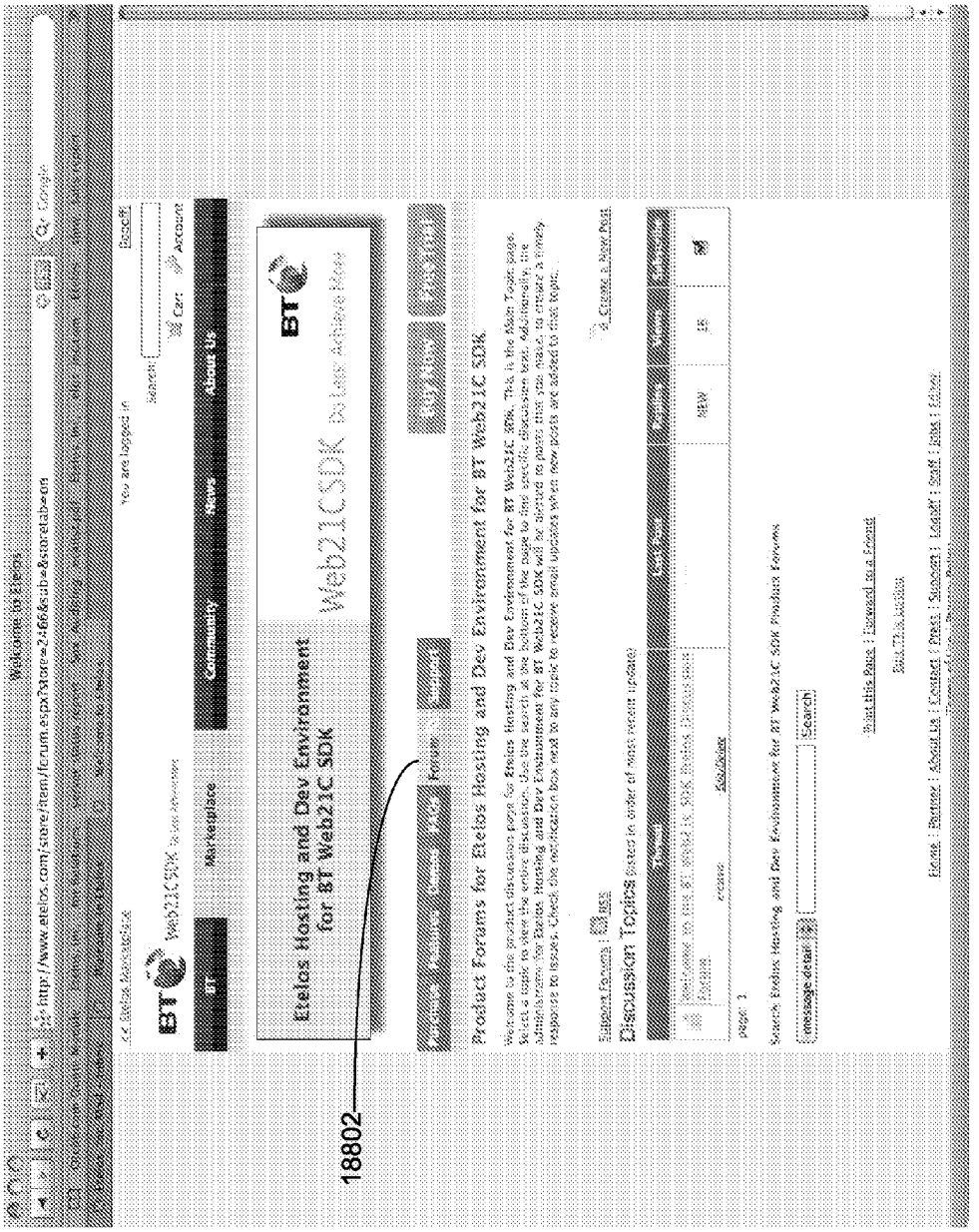

18802
18800
Etelos Hosting and Dev Environment for BT Web21C SDK
Web21C SDK
Product Forums for Etelos Hosting and Dev Environment for BT Web21C SDK
Discussion Topics

Figure 188

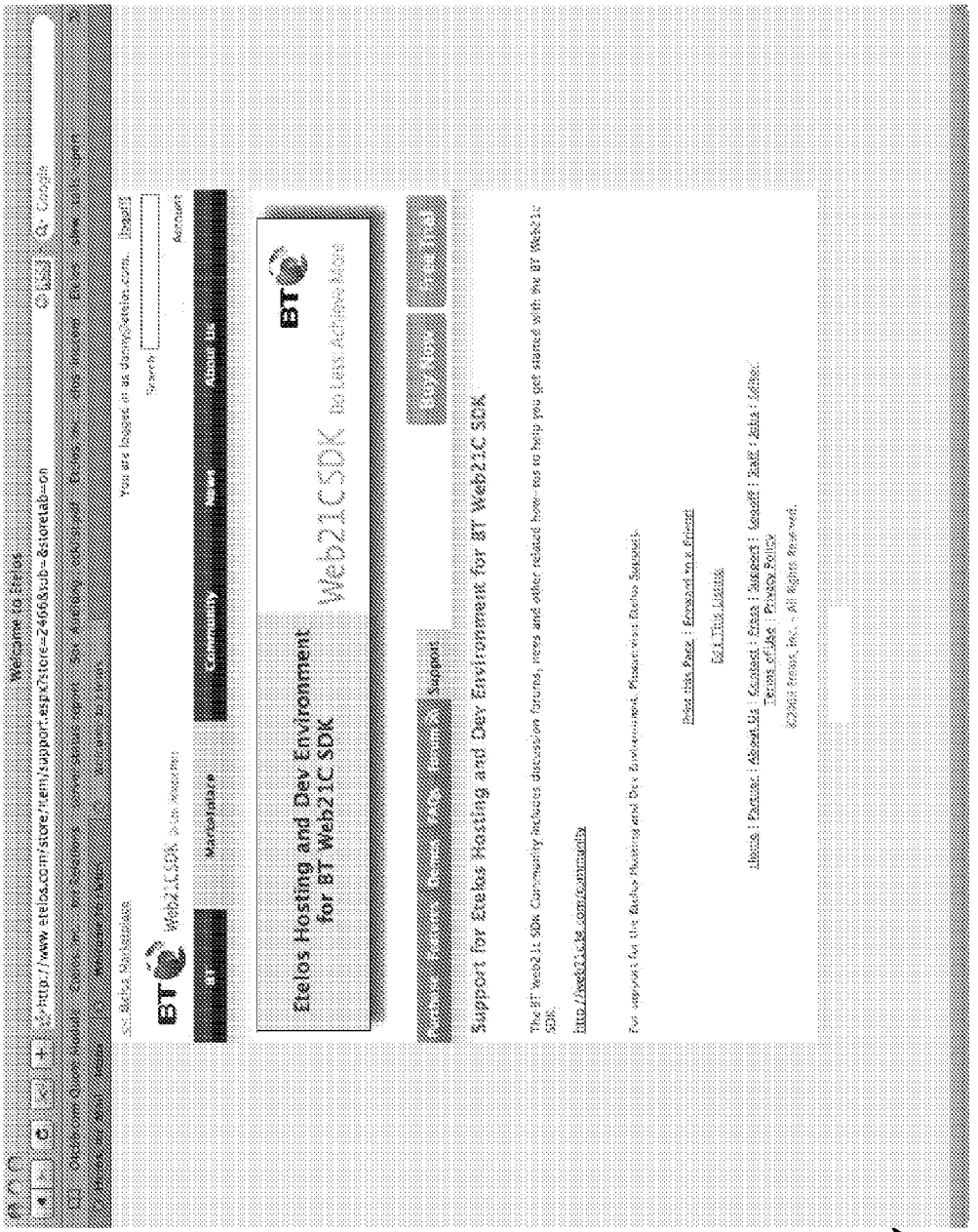

Welcome to Etelos
Etelos Hosting and Dev Environment for BT Web21C SDK
Web21C SDK
Marketplace
Community
Apps
About Us
Support for Etelos Hosting and Dev Environment for BT Web21C SDK
18900

Figure 189

SOFTWARE LICENSING AND ENFORCEMENT SYSTEM

RELATED APPLICATIONS

[0001] This application claims the benefit of and priority to U.S. Provisional Patent Application Ser. No. 60/962,877 filed on Jul. 31, 2007, the disclosure of which is hereby incorporated by reference in its entirety. This application is a continuation-in-part of U.S. patent application Ser. No. 12/102,854, "System And Method For Resolving Conflicts Between An Offline Web-Based Application And An Online Web-Based Application" filed on Apr. 14, 2008, which application is incorporated by reference herein in its entirety.

[0002] This application is related to U.S. patent application Ser. No. 12/102,848, "System And Method For Synchronizing An Offline Web-Based Application With An Online Web-Based Application" filed on Apr. 14, 2008, which application is incorporated by reference herein in its entirety. This application is related to U.S. patent application Ser. No. 12/102,842, "System And Method For Running A Web-Based Application While Offline" filed on Apr. 14, 2008, which application is incorporated by reference herein in its entirety. This application is related to U.S. patent application Ser. No. _____, "System and Method for Synchronizing Applications" filed on the same date as this application, (Attorney Docket Number 069904-5004), which application is incorporated by reference herein in its entirety. This application is related to U.S. patent application Ser. No. _____, "Framework for Synchronizing Applications" filed on the same date as this application, (Attorney Docket Number 069904-5005), which application is incorporated by reference herein in its entirety. This application is related to U.S. patent application Ser. No. _____, "Software Marketplace and Distribution System" filed on the same date as this application, (Attorney Docket Number 069904-5006), which application is incorporated by reference herein in its entirety.

TECHNICAL FIELD

[0003] The disclosed embodiments relate generally to licensing and enforcement of software applications.

BACKGROUND

[0004] It is often difficult for developers or vendors of software applications, particularly small software applications, to economically market and license their applications. Furthermore, it is often difficult to enforce a license for a small software application, because it may not be cost effective to pursue a violator of the license. It would be desirable to have a system for marketing, licensing and enforcing of software applications that permits a software developer or vendor to concentrate on creating software applications without the overhead of managing the business aspects of marketing and licensing.

SUMMARY

[0005] An embodiment of the present application relates to a marketplace for software applications where, once licensed, the software applications are hosted at a user account.

[0006] The present application describes some embodiments of a software marketplace whereby software vendors can easily upload and license software applications and receive revenue in return. Among other advantages, this frees software vendors from the need to manage financial and legal issues associated with licensing software applications to large numbers of users. In one embodiment, the software marketplace is associated with a software platform provider (in one example, Etelos) and the software vendors develop software applications for this software platform. This arrangement benefits both the software vendor (who can concentrate on writing applications and receiving revenue for them) and the software platform provider (who has a large number of developers supporting their software platform.

[0007] This arrangement is particularly attractive to vendors of small software applications, where the revenue per licensed application is small, and the number of licensees is high. It may not be cost effective or time effective for the software vendor to engage with large numbers of small payments and licensees, particularly when the licensees may be spread around geographically, in different time zones, use different currencies, etc. By combining ease of use, tight integration, and transparent billing and licensing for the vendors' software applications, the software platform provider can provide an attractive service for its customers

[0008] As the number of software vendors supporting the software platform increases, the software vendors may provide custom application development to customers of the software platform. In some embodiments, the software platform provider can monitor this process and ensure quality. In some embodiments, customers of the software platform may place jobs (i.e., custom software specifications) out for bid, where developers bid on the work. A software customer may specify a bid based on a combination of cost, quality, delivery time, and other factors.

[0009] Software vendors are commonly concerned about the overhead of licensing their software applications, and about enforcing their software licenses. Some embodiments enable software vendors to specify a set of license terms (e.g., commonly used license types such as open source, proprietary, executable only, source code license, etc.) for a software application, and prevent licensees of the software application from misusing the software application outside the terms of the license.

[0010] Some embodiments provide a method for detecting changes made to a first data set in a plurality of data sets, and synchronizing at least a first subset of the changes to a data framework that facilitates data synchronization between the plurality of data sets.

[0011] In some embodiments, at least a second subset of the synchronized changes from the data framework to a second data set in the plurality of data sets is synchronized.

[0012] In some embodiments, at least a third subset of the synchronized changes from the data framework to a second data framework is synchronized.

[0013] Some embodiments provide a computer readable storage medium storing one or more programs configured for execution by a computer, the one or more programs including instructions for detecting changes made to a first data set in a plurality of data sets, and synchronizing at least a first subset of the changes to a data framework that facilitates data synchronization between the plurality of data sets.

[0014] Some embodiments provide a system including one or more processors, memory, and one or more programs stored in the memory, the one or more programs comprising instructions to: detect changes made to a first data set in a plurality of data sets; and synchronize at least a first subset of the changes to a data framework that facilitates data synchronization between the plurality of data sets.

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 180

[0015] Some embodiments provide a method for identifying a first data set in a plurality of data sets that is to be synchronized with a data framework, determining a mapping between one or more data fields in a data structure for the first data set and one or more data fields in a data structure for the data framework, and generating synchronization rules for the first data set based on the determined mapping.

[0016] Some embodiments provide a computer readable storage medium storing one or more programs configured for execution by a computer, the one or more programs including instructions for identifying a first data set in a plurality of data sets that is to be synchronized with a data framework, determining a mapping between one or more data fields in a data structure for the first data set and one or more data fields in a data structure for the data framework, and generating synchronization rules for the first data set based on the determined mapping.

[0017] Some embodiments provide a system including one or more processors, memory, and one or more programs stored in the memory, the one or more programs comprising instructions to: identify a first data set in a plurality of data sets that is to be synchronized with a data framework, determine a mapping between one or more data fields in a data structure for the first data set and one or more data fields in a data structure for the data framework, and generate synchronization rules for the first data set based on the determined mapping.

[0018] Some embodiments provide a method for detecting changes made to a first data set for a first web-based application in an account, identifying at least a second data set for a second web-based application in the account, wherein the second data set includes at least a subset of the data included in the first data set, and applying synchronization rules to synchronize the second data set with the first data set.

[0019] Some embodiments provide a computer readable storage medium storing one or more programs configured for execution by a computer, the one or more programs including instructions for detecting changes made to a first data set for a first web-based application in an account, identifying at least a second data set for a second web-based application in the account, wherein the second data set includes at least a subset of the data included in the first data set, and applying synchronization rules to synchronize the second data set with the first data set.

[0020] Some embodiments provide a system including one or more processors, memory, and one or more programs stored in the memory, the one or more programs comprising instructions to: detect changes made to a first data set for a first web-based application in an account, identify at least a second data set for a second web-based application in the account, wherein the second data set includes at least a subset of the data included in the first data set, and apply synchronization rules to synchronize the second data set with the first data set.

[0021] Some embodiments provide a computer system including one or more processors, memory, and one or more programs and data structures stored in the memory, the one or more programs and data structures including: an application data structure configured to store data and program files for a web-based application, an account data structure configured to store one or more instances of the application data structure for an account, wherein an instance of the application data structure corresponds to an instance of a web-based application, and a synchronization module configured to synchronize data between web-based applications within an account based on synchronization rules.

[0022] Some embodiments provide a computer readable storage medium storing one or more programs and data structures configured for execution by a computer, the one or more data structures including: an application data structure configured to store data and program files for a web-based application, and an account data structure configured to store one or more instances of the application data structure for an account, wherein an instance of the application data structure corresponds to an instance of a web-based application. The one or more programs include instructions for synchronizing data between web-based applications within an account based on synchronization rules.

[0023] In accordance with some embodiments, a computer-implemented method is performed at a system. The computer-implemented method includes at one or more servers hosting a marketplace application: receiving from a vendor a software application for distribution; associating license terms with the software application; making the software application available for distribution through the marketplace application; and deploying the software application to one or more user accounts on one or more hosting servers, in accordance with the license terms.

[0024] In accordance with some embodiments, a computer-implemented method is performed at a system. The computer-implemented method includes at one or more servers hosting a marketplace application: receiving from a vendor a software application for distribution; generating license terms in response to a selection by the vendor from options provided by the marketplace application; associating the license terms with the software application; and making the software application available for distribution through the marketplace application, in accordance with the license terms.

[0025] In accordance with some embodiments, a computer-implemented method is performed at a system. The computer-implemented method includes at one or more servers hosting a marketplace application, in response to a request from a syndicated server to distribute a software application from the marketplace: identifying one or more user accounts associated with the request; verifying that the one or more user accounts has permission to use the software application; and deploying the software application to the one or more user accounts, in accordance with license terms associated with the software application.

[0026] In accordance with some embodiments, a computer-implemented method is performed at a system. The computer-implemented method includes at one or more servers hosting a marketplace application: making a software application available for distribution through the marketplace application; receiving a user request to license the software application; and providing the software application for deployment to one or more user accounts, hosted on one or more servers.

[0027] In accordance with some embodiments, a system for distributing software applications is described. The system comprises one or more processors, memory, and one or more programs stored in the memory. The one or more programs comprise instructions for implementing: a program module configured to provide a software application for distribution in response to an access request from a user; a program module configured to receive and deploy the software application from the marketplace module to an account on one or

more servers; and a program module configured to provide at least one or more user accounts, from which the user accesses the software application.

[0028] In accordance with some embodiments, a server system comprises one or more processors, memory, and one or more programs stored in the memory. The one or more programs comprise instructions for at one or more servers hosting a marketplace application: receiving from a vendor a software application for distribution; associating license terms with the software application; making the software application available for distribution through the marketplace application; and deploying the software application to one or more user accounts on one or more hosting servers, in accordance with the license terms.

[0029] In accordance with some embodiments, a computer readable storage medium stores one or more programs configured for execution by a computer. The one or more programs comprise instructions to, at one or more servers hosting a marketplace application: receive from a vendor a software application for distribution; associate license terms with the software application; make the software application available for distribution through the marketplace application; and deploy the software application to one or more user accounts on one or more hosting servers, in accordance with the license terms.

[0030] In accordance with some embodiments, a server system comprises one or more processors, memory, and one or more programs stored in the memory. The one or more programs comprise instructions for, at one or more servers hosting a marketplace application: receiving from a vendor a software application for distribution; generating license terms in response to a selection by the vendor from options provided by the marketplace application; associating the license terms with the software application; and making the software application available for distribution through the marketplace application, in accordance with the license terms.

[0031] In accordance with some embodiments, a computer readable storage medium stores one or more programs configured for execution by a computer. The one or more programs comprise instructions for, at one or more servers hosting a marketplace application: receiving from a vendor a software application for distribution; generating license terms in response to a selection by the vendor from options provided by the marketplace application; associating the license terms with the software application; and making the software application available for distribution through the marketplace application, in accordance with the license terms.

[0032] In accordance with some embodiments, a server system comprises one or more processors, memory, and one or more programs stored in the memory. The one or more programs comprise instructions for, at one or more marketplace servers hosting a marketplace application, in response to a request from a syndicated server to distribute a software application from the marketplace: identifying one or more user accounts associated with the request; verifying that the one or more user accounts has permission to use the software application; and deploying the software application to the one or more user accounts, in accordance with license terms associated with the software application.

[0033] In accordance with some embodiments, a computer readable storage medium stores one or more programs configured for execution by a computer. The one or more programs comprise instructions for, at one or more marketplace servers hosting a marketplace application, in response to a request from a syndicated server to distribute a software application from the marketplace: identifying one or more user accounts associated with the request; verifying that the one or more user accounts has permission to use the software application; and deploying the software application to the one or more user accounts, in accordance with license terms associated with the software application.

[0034] In accordance with some embodiments, a server system comprises one or more processors, memory, and one or more programs stored in the memory. The one or more programs comprise instructions for at one or more servers hosting a marketplace application: making a software application available for distribution through the marketplace application; receiving a user request to license the software application; and providing the software application for deployment to one or more user accounts, hosted on one or more servers.

[0035] In accordance with some embodiments, a computer readable storage medium stores one or more programs configured for execution by a computer. The one or more programs comprise instructions for at one or more servers hosting a marketplace application: making a software application available for distribution through the marketplace application; receiving a user request to license the software application; and providing the software application for deployment to one or more user accounts, hosted on one or more servers.

BRIEF DESCRIPTION OF THE DRAWINGS

[0036] FIG. **1** presents a block diagram of a network, according to embodiments of the present invention.

[0037] FIG. **2** presents a block diagram of a network, according to embodiments of the present invention.

[0038] FIG. **3** presents a block diagram of an application database, according to embodiments of the present invention.

[0039] FIG. **4** presents a block diagram of a user database, according to embodiments of the present invention.

[0040] FIG. **5** presents a block diagram of a synchronization engine on an application server, according to embodiments of the present invention.

[0041] FIG. **6** presents a block diagram of a synchronization engine on a client computer system, according to embodiments of the present invention.

[0042] FIG. **7** presents a block diagram of an application server, according to embodiments of the present invention.

[0043] FIG. **8** presents a block diagram of a client, according to embodiments of the present invention.

[0044] FIG. **9** presents a block diagram illustrating exemplary group memberships for users for a given web-based application, according to embodiments of the present invention.

[0045] FIG. **10** presents a block diagram of an exemplary AOP framework that provides offline access to a web-based application, according to embodiments of the present invention.

[0046] FIG. **11** illustrates an exemplary process of using an AOP framework to run a web-based application while offline, according to embodiments of the present invention.

[0047] FIG. **12** presents a block diagram of an exemplary process of installing an instance of an AOP framework on a client computer system, according to embodiments of the present invention.

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 182

[0048] FIG. **13** illustrates an exemplary process of installing an instance of an AOP framework on a client computer system, according to embodiments of the present invention.

[0049] FIG. **14** presents a block diagram of an exemplary process of initializing an AOP account on a client computer system, according to embodiments of the present invention.

[0050] FIG. **15** illustrates an exemplary process of initializing an AOP account on a client computer system, according to embodiments of the present invention.

[0051] FIG. **16** presents a block diagram of an exemplary process of installing a web-based application on a client computer system, according to embodiments of the present invention.

[0052] FIG. **17** illustrates an exemplary process of installing a web-based application on a client computer system, according to embodiments of the present invention.

[0053] FIG. **18** presents a block diagram of an exemplary process of performing an initial data synchronization for a web-based application on a client computer system, according to embodiments of the present invention.

[0054] FIG. **19** illustrates an exemplary process of performing an initial data synchronization for a web-based application on a client computer system, according to embodiments of the present invention.

[0055] FIG. **20** presents a block diagram of an exemplary process of using the AOP framework on a client computer system, according to embodiments of the present invention.

[0056] FIG. **21** illustrates an exemplary process of using the AOP framework on a client computer system, according to embodiments of the present invention.

[0057] FIG. **22** presents a block diagram of an exemplary process for a client computer system determining how to access a web-based application, according to embodiments of the present invention.

[0058] FIG. **23** illustrates an exemplary process for a client computer system determining how to access a web-based application, according to embodiments of the present invention.

[0059] FIG. **24** presents a block diagram of an exemplary process for synchronizing a web-based application on a client computer system with a web-based application on an application server, according to embodiments of the present invention.

[0060] FIG. **25** illustrates an exemplary process for synchronizing a web-based application on a client computer system with a web-based application on an application server, according to embodiments of the present invention.

[0061] FIG. **26** presents a block diagram of an exemplary process for synchronizing a web-based application on a client computer system with a web-based application on an application server, according to embodiments of the present invention.

[0062] FIG. **27**A illustrates an exemplary process for synchronizing a web-based application on a client computer system with a web-based application on an application server, according to embodiments of the present invention.

[0063] FIG. **27**B continues the process illustrated in FIG. **27**A, according to embodiments of the present invention.

[0064] FIG. **27**C continues the process illustrated in FIG. **27**B, according to embodiments of the present invention.

[0065] FIG. **27**D continues the process illustrated in FIG. **27**C, according to embodiments of the present invention.

[0066] FIG. **27**E continues the process illustrated in FIG. **27**D, according to embodiments of the present invention.

[0067] FIG. **27**F continues the process illustrated in FIG. **27**E, according to embodiments of the present invention.

[0068] FIG. **28** presents a block diagram of an exemplary process for resolving conflicts for web-based applications that use auto-incrementing identifiers, according to embodiments of the present invention.

[0069] FIG. **29** illustrates an exemplary process for resolving conflicts for web-based applications that use auto-incrementing identifiers, according to embodiments of the present invention.

[0070] FIG. **30** presents a flowchart of an exemplary process for providing access to a web-based application while offline, according to embodiments of the present invention.

[0071] FIG. **31** presents a flowchart of an exemplary process for synchronizing a web-based application on a client computer system with a web-based application on an application server, according to embodiments of the present invention.

[0072] FIG. **32** presents a flowchart of an exemplary process for synchronizing a web-based application on a client computer system with a web-based application on an application server, according to embodiments of the present invention.

[0073] FIG. **33** presents a flowchart of an exemplary process for resolving conflicts between a web-based application on a client computer system and a web-based application on an application server, according to embodiments of the present invention.

[0074] FIG. **34** presents a flowchart of an exemplary process for resolving conflicts between a web-based application on a client computer system and a web-based application on an application server which uses automatically incrementing identifiers for database records, according to embodiments of the present invention.

[0075] FIG. **35** presents a flowchart of an exemplary process for providing access to a web-based application while offline, according to embodiments of the present invention.

[0076] FIG. **36** presents a block diagram illustrating an exemplary user interface for creating synchronization rules, according to embodiments of the present invention.

[0077] FIG. **37** presents a block diagram illustrating an exemplary user interface for generating auto-incrementing identifiers, according to embodiments of the present invention.

[0078] FIG. **38** presents a block diagram illustrating an exemplary process for creating synchronization rules, according to embodiments of the present invention.

[0079] FIG. **39** presents a flowchart of an exemplary process for creating synchronization rules, according to embodiments of the present invention.

[0080] FIG. **40** presents a block diagram of an exemplary foreign key mapping, according to embodiments of the present invention.

[0081] FIG. **41** presents a flowchart of an exemplary process for creating a foreign key mapping, according to embodiments of the present invention.

[0082] FIG. **44** is a block diagram illustrating deploying applications to a user account, according to some embodiments.

[0083] FIG. **45** is a block diagram illustrating managing licenses in a multi-tenancy environment, according to some embodiments.

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 183

[0084] FIG. **46** is a block diagram illustrating a web application marketplace and hosting infrastructure, according to some embodiments.

[0085] FIG. **47** is a flow diagram illustrating a process for selecting and deploying a software application in a user account, according to some embodiments.

[0086] FIG. **48** is a block diagram illustrating a marketplace server and hosting infrastructure, according to some embodiments.

[0087] FIG. **49** is a block diagram illustrating licensing options in a software distribution marketplace, according to some embodiments.

[0088] FIG. **50** is a block diagram illustrating dynamic billing in a software distribution marketplace, according to some embodiments.

[0089] FIG. **51** is a block diagram illustrating an application manager and repository in a software distribution marketplace, according to some embodiments.

[0090] FIG. **52** is a block diagram illustrating syndicated deployment across a network in a software distribution marketplace, according to some embodiments.

[0091] FIG. **53** is a block diagram illustrating a packager in a software distribution marketplace, according to some embodiments.

[0092] FIG. **54** is a block diagram illustrating an alternative embodiment of a packager in a software distribution marketplace, according to some embodiments.

[0093] FIG. **55** is a block diagram illustrating an alternative embodiment of a packager in a software distribution marketplace, according to some embodiments.

[0094] FIG. **56** is a block diagram illustrating licensing a software application in a software distribution marketplace, according to some embodiments.

[0095] FIG. **57** is a block diagram illustrating providing security for software applications, deployed to a user account, in a software distribution marketplace, according to some embodiments.

[0096] FIG. **58** is a block diagram illustrating multiple logons to a software application in a software distribution marketplace, according to some embodiments.

[0097] FIG. **59** is a block diagram illustrating a user access control interface, in a software distribution marketplace, according to some embodiments.

[0098] FIG. **60** is a block diagram illustrating a user access control interface, in a software distribution marketplace, according to some embodiments.

[0099] FIG. **61** is a system block diagram illustrating a server hosting a software marketplace and licensing system, according to some embodiments.

[0100] FIG. **62** is a system block diagram illustrating a client interfacing with a software marketplace and licensing system, according to some embodiments.

[0101] FIG. **63** is a block diagram illustrating an exemplary application framework, according to some embodiments.

[0102] FIG. **64**A is a block diagram illustrating exemplary components of an account, according to some embodiments.

[0103] FIG. **64**B is a flow diagram of an exemplary process for creating an account and installing applications into the account, according to some embodiments.

[0104] FIG. **65** is a block diagram of a server and a client, according to some embodiments.

[0105] FIG. **66** is a flow diagram of an exemplary process for synchronizing applications, according to some embodiments.

[0106] FIG. **67** is a block diagram illustrating an exemplary process for synchronizing applications, according to some embodiments.

[0107] FIG. **68** is a block diagram illustrating an exemplary process for handling parent-child relationships during a synchronization process, according to some embodiments.

[0108] FIG. **69** is a flow diagram of an exemplary process for translating owner IDs, according to some embodiments.

[0109] FIG. **70** is a block diagram illustrating an exemplary process for merging users between applications, according to some embodiments.

[0110] FIG. **71** is a flow diagram of an exemplary process for merging users between applications, according to some embodiments.

[0111] FIG. **72** presents a block diagram of an exemplary server, according to some embodiments.

[0112] FIG. **73** presents a block diagram of an exemplary client computer system, according to some embodiments.

[0113] FIG. **74** presents a block diagram illustrating an exemplary application synchronization module, according to some embodiments.

[0114] FIG. **75** presents exemplary synchronization data structures, according to some embodiments.

[0115] FIG. **76** is a flow diagram of an exemplary process for synchronizing applications, according to some embodiments.

[0116] FIG. **77** is a flow diagram of an exemplary process for generating synchronization rules that are used to synchronize applications, according to some embodiments.

[0117] FIG. **78** is a flow diagram of an exemplary process for synchronizing applications, according to some embodiments.

[0118] FIG. **79** presents exemplary synchronization rules data structures, according to some embodiments.

[0119] FIG. **80** is a flow diagram of a process for distributing a software application, according to some embodiments.

[0120] FIG. **81** is a flow diagram of a process for distributing a software application, according to some embodiments.

[0121] FIG. **82** is a flow diagram of a process for licensing a software application, according to some embodiments.

[0122] FIG. **83** is a flow diagram of a process for distributing a software application, according to some embodiments.

[0123] FIG. **84** is a flow diagram of a process for syndicated deployment of a software application, according to some embodiments.

[0124] FIG. **85** is a flow diagram of a process for licensing and receiving payment for a software application, according to some embodiments.

[0125] FIG. **101** is an exemplary screenshot **10100** of an account user management interface.

[0126] FIG. **102** is an exemplary screenshot **10200** of an application packager.

[0127] FIG. **103** is an exemplary screenshot **10300** of an application packager.

[0128] FIG. **104** is an exemplary screenshot **10400** of an application packager.

[0129] FIG. **105** is an exemplary screenshot **10500** of an application packager.

[0130] FIG. **106** is an exemplary screenshot **10600** of an application packager.

[0131] FIG. **107** is exemplary screenshot **10700** of an application packager.

[0132] FIG. **108** is an exemplary screenshot **10800** of an application packager.

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 184

[0133] FIG. **109** is an exemplary screenshot **10900** of an application packager.
[0134] FIG. **110** is an exemplary screenshot **11000** of an application packager.
[0135] FIG. **111** is an exemplary screenshot **11100** of an application packager.
[0136] FIG. **112** is an exemplary screenshot **11200** of an application packager.
[0137] FIG. **113** is an exemplary screenshot **11300** of an application packager.
[0138] FIG. **114** is an exemplary screenshot **11400** of an account information details screen.
[0139] FIG. **115** is an exemplary screenshot **11500** of a development environment.
[0140] FIG. **116** is an exemplary screenshot **11600** of an application packager.
[0141] FIG. **117** is an exemplary screenshot **11700** of an AOP sync rules manager.
[0142] FIG. **118** is an exemplary screenshot **11800** of an AOP sync rules manager.
[0143] FIG. **119** is an exemplary screenshot **11900** of an AOP sync rules manager.
[0144] FIG. **120** is an exemplary screenshot **12000** of an AOP sync rules manager.
[0145] FIG. **121** is an exemplary screenshot **12100** of an AOP sync rules manager.
[0146] FIG. **122** is an exemplary screenshot **12200** of an integration application group manager.
[0147] FIG. **123** is an exemplary screenshot **12300** of a sync rules manager.
[0148] FIG. **124** is an exemplary screenshot **12400** of a sync rules manager.
[0149] FIG. **125** is an exemplary screenshot **12500** of a sync rules manager.
[0150] FIG. **126** is an exemplary screenshot **12600** of a sync rules manager.
[0151] FIG. **127** is an exemplary screenshot **12700** of a sync rules manager.
[0152] FIG. **128** is an exemplary screenshot **12800** of a sync rules manager.
[0153] FIG. **129** is an exemplary screenshot **12900** of a support page.
[0154] FIG. **130** is an exemplary screenshot **13000** of a product/service basic information page.
[0155] FIG. **131** is an exemplary screenshot **13100** of a staging application page.
[0156] FIG. **132** is an exemplary screenshot **13200** of an edit licensing page.
[0157] FIG. **133** is an exemplary screenshot **13300** of a setup marketing information page.
[0158] FIG. **134** is an exemplary screenshot **13400** of a setup marketing full page.
[0159] FIG. **135** is an exemplary screenshot **13500** of a setup features full page.
[0160] FIG. **136** is an exemplary screenshot **13600** of a product demo page.
[0161] FIG. **137** is an exemplary screenshot **13700** of an "Add a Blog" feed page.
[0162] FIG. **138** is an exemplary screenshot **13800** of a setup frequently asked questions (FAQs) page.
[0163] FIG. **139** is an exemplary screenshot **13900** of a setup getting started application page.
[0164] FIG. **140** is an exemplary screenshot **14000** of a support page.
[0165] FIG. **141** is an exemplary screenshot **14100** of an "About us" page.
[0166] FIG. **142** is an exemplary screenshot **14200** of a product upgrade page.
[0167] FIG. **143** is an exemplary screenshot **14300** of a "my store style" page.
[0168] FIG. **144** is an exemplary screenshot **14400** of a pricing grid and purchase/license link setup page.
[0169] FIG. **145** is an exemplary screenshot **14500** of a web services "Post Data" setup page.
[0170] FIG. **146** is an exemplary screenshot **14600** of a support page.
[0171] FIG. **147** is an exemplary screenshot **14700** of a support page.
[0172] FIG. **148** is an exemplary screenshot **14800** of a support page.
[0173] FIG. **149** is an exemplary screenshot **14900** of a support page.
[0174] FIG. **150** is an exemplary screenshot **15000** of a store product list.
[0175] FIG. **151** is an exemplary screenshot **15100** of a store listing report.
[0176] FIG. **152** is an exemplary screenshot **15200** of a store listing report.
[0177] FIG. **153** is a screenshot of an exemplary screenshot **15300** of a transactions report.
[0178] FIG. **154** is an exemplary screenshot **15400** of a support page.
[0179] FIG. **155** is an exemplary screenshot **15500** of a marketplace page.
[0180] FIG. **156** is an exemplary screenshot **15600** of a hosting and development environment.
[0181] FIG. **157** is an exemplary screenshot **15700** of a marketplace page.
[0182] FIG. **158** is an exemplary screenshot **15800** of a licensing page.
[0183] FIG. **159** is an exemplary screenshot **15900** of a support page.
[0184] FIG. **160** is an exemplary screenshot **16000** of a shopping cart page.
[0185] FIG. **161** is an exemplary screenshot **16100** of a CRM test listing page.
[0186] FIG. **162** is an exemplary screenshot **16200** of a developer toolkit page.
[0187] FIG. **163** is an exemplary screenshot **16300** of a support page.
[0188] FIG. **164** is an exemplary screen shot **16400** from the bottom portion of the screenshot **16300**.
[0189] FIG. **165** is an exemplary screenshot **16500** of a support page.
[0190] FIG. **166** is an exemplary screenshot **16600** of an installation page.
[0191] FIG. **167** is an exemplary screenshot **16700** of an installation processing page.
[0192] FIG. **168** is an exemplary screenshot **16800** of an installation processing page.
[0193] FIG. **169** is an exemplary screenshot **16900** of a support page.
[0194] FIG. **170** is an exemplary screenshot **17000** of a support page.
[0195] FIG. **171** is an exemplary screenshot **17100** of a marketplace page.
[0196] FIG. **172** is an exemplary screenshot **17200** of a marketplace page.

[0197] FIG. **173** is an exemplary screenshot **17300** of a support page.
[0198] FIG. **174** is an exemplary screenshot **17400** of a support page.
[0199] FIG. **175** is an exemplary screenshot **17500** of a support page.
[0200] FIG. **176** is an exemplary screenshot **17600** of a support page.
[0201] FIG. **177** is an exemplary screenshot **17700** of a support page.
[0202] FIG. **178** is an exemplary screenshot **17800** of a support page.
[0203] FIG. **179** is an exemplary screenshot **17900** of a support page.
[0204] FIG. **180** is an exemplary screenshot **18000** of a support page.
[0205] FIG. **181** is an exemplary screenshot **18100** of a support page.
[0206] FIG. **182** is an exemplary screenshot **18200** of a marketplace homepage.
[0207] FIG. **183** is an exemplary screenshot **18300** of a marketplace page.
[0208] FIG. **184** is an exemplary screenshot **18400** of a marketplace page.
[0209] FIG. **185** is an exemplary screenshot **18500** of a marketplace page.
[0210] FIG. **186** is an exemplary screenshot **18600** of a marketplace page.
[0211] FIG. **187** is an exemplary screenshot **18700** of a marketplace page.
[0212] FIG. **188** is an exemplary screenshot **18800** of a marketplace page.
[0213] FIG. **189** is an exemplary screenshot **18900** of a marketplace page.
[0214] Like reference numerals refer to corresponding parts throughout the drawings.

DESCRIPTION OF EMBODIMENTS

Definitions

[0215] A Remote Procedure Call (RPC) is a programming interface that allows one program to use the services of another program in a remote machine. The calling program sends a message and data to the remote program, which is executed, and results are passed back to the calling program. Note that RPC refers to XML RPC.
[0216] A virtual host (vhost) is a server that includes multiple web sites, each with its own domain name. A <virtualhost> . . . </virtualhost> is an Apache HTTP server directive (instruction) that maps domain names to different directories (and other instructions) on the filesystem. vhosts can be used to define the boundaries of an application. Each web-based application in an account on the client computer system must have at least one virtual host in Apache. Note that there can be more than one vhost, all pointing to the same shared directory/config. Also note that other web servers have similar functions as the Apache virtualhost directive.
[0217] Vmap is a virtual database query map. A vmap is a variable map, or a field map. It maps variables (columns, fields) in one database to variables (columns, fields) in another DB.
[0218] LAMP is a solution stack of software that is used to run dynamic web sites. The LAMP solution stack typically comprises open source software. The LAMP stack can refer to a suite of software that includes LINUX, Apache HTTP server, MySQL and/or PostgreSQL, Perl, Python, Ruby, Ruby on Rails, Apache Tomcat, and/or PHP.
[0219] A solution stack is a set of software subsystems or components that are required to deliver a specified solution (e.g., product or service).
[0220] A web application or a web-based application is an application that can be accessed through the web over a network (e.g., the Internet, etc.). Web-based applications typically generate dynamic content. Dynamic content is content that can be generated when a request is received from a user. For example, a user may request scores for one or more sporting events. These scores can be retrieved and a page listing the scores can be displayed to the user. Alternatively, dynamic content can be periodically generated and cached. When a user requests the dynamic content, the cached version is displayed to the user. Web-based applications do not require distribution because they are typically hosted on an application server. Users who desire to use a web-based application can use a browser to access the application server hosting the web-based application. A client-server model requires a specialized client program that serves as a user interface to the server and that must be installed on each computer system that is to access the server application. Any upgrades to the server application typically require an update to the client application. In contrast, web-based applications use a browser engine, such as found in a web browser engine, as an interface to the application server. In general, each page delivered to the browser is a static document; however, the static document is typically composed of a number of dynamic elements that are embedded into the document prior to being delivered to the browser. Web-based applications are typically structured as a three-tier application with a browser engine at the first tier, an engine that can process dynamic content (e.g., scripting languages, etc.) in the second tier, and a database in the third tier.

Overview

[0221] Presently, in order to receive the full functionality of a web-based application, a client computer system must be able to communicate with an application server hosting the web-based application. Although some features of a web-based application may be available while the client computer system is not connected to an application server hosting the web-based application, other features of the application server may not function properly or may not function at all if the client computer system is not connected to the application server. For example, consider a user who wants to update a contact in a web-based address book. Updating a contact in a web-based address book typically requires updating data records in a database or some other mechanism for storing and managing data (e.g., files). Typically, the database is part of the application server or accessible to the application server through a network connection. Thus, in order to update a contact in the web-based address book, the client computer system for the user must be connected to the application server hosting the web-based address book so that the update to the contact can be made to in the database.
[0222] Thus, in some embodiments, a client computer system is loaded with a framework that allows the client computer system to access web-based applications locally without requiring a network connection to an application server hosting the web-based application. This framework is referred to as the "Applications on a Plane" (AOP) framework

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 186

in this specification. This framework is useful for at least traveling salespeople, drivers, business travelers (e.g., on airplanes), etc. In these embodiments, while working on the application locally, data can be added, modified, and changed without the need to be connected to the application server hosting the web-based application. The AOP framework tracks the changes and when the network connection between the client computer system and the application server is reestablished, the changes can be synchronized between the client computer system and the application server.

[0223] In some embodiments, the AOP framework can include an application server and an account management system. In some embodiments, the application server is the Etelos Application Server (EAS). In some embodiments, the account management system is the Etelos Management System (EMS). This specification may use the terms EAS and EMS in the generic sense to refer to an application server and an account management system, respectively.

[0224] In some embodiments, AOP framework installations can be split into two steps: installation of an open source stack (e.g., Apache HTTP server, PHP, PostgreSQL, MySQL, Python, etc.) and installation of the AOP framework (e.g., the EAS and EMS installation). Other software stacks can be used. For example, WAMP (Windows, Apache HTTP server, MySQL, PHP) and/or LAPP (Linux, Apache HTTP server, PostgreSQL, PHP).

[0225] In some embodiments, AOP synchronization is managed at two levels. A user-based synchronization enables the user to synchronize information that only the user is allowed to see based upon a set of user-based synchronization rules. A group administrative synchronization synchronizes all changes in the web-based application for a user that a user is allowed to see because of group administrative permissions.

[0226] In some embodiments, the AOP framework allows existing web-based applications to support offline use on a client computer system (e.g., without a connection to an application server that hosts the web-based application) without changing the architecture and code for the web-based application. For example, if a developer builds a web-based application based on the Linux, Apache HTTP server, MySQL, and PHP (LAMP) web development environment, the web-based application can be used in the AOP framework with little or no code changes. A developer can then use an administrative tool to create rules for how the web-based application is to be synchronized with client computer systems that have made changes to the web-based application while offline. Note that in prior art systems, a developer must re-architect (e.g., changing the data model) and/or recode the web-based application to be able to run on a client computer system without a network connection to an application server.

[0227] In some embodiments, the AOP framework assigns a local domain extension to a universal resource locator (URL) for a web-based application so that a user can access the web-based application on the client computer system instead of on the application server. In doing so, a user can choose when to access the web-based application locally and when to access the web-based application on the application server. For example, the local domain extension can be an ".aop" suffix that is added to the end of the URL. If the URL is http://appl.com, the modified URL is http://appl.com.aop. If a user wants to run the web-based application on the AOP framework on a client computer system, the user can access the web-based application by entering the local URL (e.g., http://appl.com.aop). Note that the ".aop" suffix is one example of a suffix that can be appended to a URL. Any other suffix can be appended to a URL. Alternatively, the URL can be modified in other ways (e.g., completely rewriting the URL, etc.) that can indicate local access is required. In some embodiments, an entry in a hosts file is added to map the URL with the appended suffix to the local client computer system. In other embodiments, an entry in a domain naming service is added to map the URL with the appended suffix to the local client computer system.

[0228] In some embodiments, a suffix is not appended to the URL. In these embodiments, the URL itself is used to direct the browser to the local web server on the client computer system. In these embodiments, an entry in the hosts file or in a DNS server on the client computer system associates the URL with the client computer system. Note that since a client computer system starts its search for an IP address associated with the URL on the client computer system, if an IP address is found in the hosts file or a local DNS server on the client computer system, the client computer system uses this IP address regardless as to whether or not another IP address (e.g., the real IP address exists).

[0229] Allowing users to make changes to web-based applications while disconnected from an application server hosting the web-based application creates several problems including synchronization of data between the client computer system and the application server. As long as the client computer system is connected to the application server, the two systems can remain synchronized with each other by periodically communicating with each other (e.g., through polling or through triggers). However, offline use of web-based applications can cause data conflicts because users can be creating, modifying, and deleting records on different instances of the web-based application. This problem is compounded by the fact that most web-based applications use automatically incrementing database identifiers to provide a unique identifier for a given database record. For example, for a user table, the user ID column may use an automatically incrementing user ID generator. When a new user is added to the user table, the database determines the next unique user ID from the automatically incrementing user ID generator. Since, each instance of the web-based application includes the automatically incrementing user ID generator that increments the next user ID independently of the other instances of the web-based application, it is possible that the user IDs between all of the instances of the web-based application are not unique. Thus, in some embodiments, the AOP framework provides a synchronization technique that can solve the above-described problems. The synchronization technique can be separate from the web-based application so that the code for the web-based application does not need to be modified. Not modifying the code is advantageous for at least the reason that rearchitecting and recoding web-based applications can be a burdensome and time-consuming process.

[0230] Most applications that are designed for the web are designed to run on a single large piece of infrastructure with shared resources. This technique allows a "software as a service" (SaaS) provider to scale the infrastructure as needed. Since many users (e.g., companies) may be using the same database separated only by differences in primary keys, control to applications and databases are set so that security is not compromised. These security models are difficult to port to existing systems that allow use of web-based applications while disconnected from an application server. However, the

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 187

AOP framework solves these problems as described below. In some embodiments, the AOP framework enables users to see only their own data and not see data from other users. However, if a user is a part of an administrative group, the administrative user can see data for other users over which the administrative user has administrative rights.

[0231] In some embodiments, the AOP platform provides mechanisms for distribution of web-based applications through a marketplace. These mechanism can facilitate billing services (e.g., for purchases, subscriptions, and other licenses) and user management.

AOP

[0232] FIG. **1** presents a block diagram of network **100**, according to embodiments of the present invention. Network **100** includes clients **110**-A to **110**-N and application servers **130**-A to **130**-N. Clients **110**-A to clients **110**-N and application servers **130**-A to **130**-N are connected to each other through network **120**. Network **120** can include, but is not limited to, a local area network (LAN), a wide area network (WAN), the Internet, an intranet, a wireless network, a mobile network, a combination of networks, or any type of network now known or later developed. Clients **110**-A to **110**-N can not only communicate with application servers **130**-A to **130**-N through network **120**, but can also communicate with each other through network **120**. Similarly, application servers **130**-A to **130**-N can communicate with each other through network **120**.

[0233] In some embodiments, application servers **130**-A to **130**-N include one or more web-based applications. In some embodiments, a web-based application is an application that is hosted on an application server and that can be accessed by clients that are connected to the application server through a network. Although a web-based application may have a set of functionality that can be used without a network connection to the application server hosting the web-based application, a web-based application typically has another set of functionality that cannot be used by a client computer system unless the client computer system is connected to the application server hosting the web-based application. For example, the set of functionality that requires a network connection to the application server can include functionality that requires access to data stored in a database for the web-based application.

[0234] FIG. **2** presents a block diagram of network **200**, according to embodiments of the present invention. Network **200** includes clients **210**-A and **210**-B, network **120**, and application server **130**. Clients **210**-A and **210**-B may correspond to any one of clients **110**-A to **110**-N illustrated in FIG. **1**. Application server **130** may correspond to any one of the application servers **130**-A to **130**-N illustrated in FIG. **1**.

[0235] Clients **210**-A and **210**-B include browser **212**, TCP/IP communications module **220**, local application **214**, local database **216**, stack **218**, synchronization application **220**, traffic management module **222**, license authentication module **224**, copy protection and data security module **226**. Browser **212** can include any application that can access data and/or services on a remote computer system through a network (e.g., network **120**). In some embodiments, browser **212** is a web browser. TCP/IP communications module **220** provides procedures and routines that allow clients **210**-A and **210**-B to communicate with other computer system through network **120** using the TCP/IP protocol.

[0236] Local application **214** can include any application that can be run on a client. In some embodiments, local application **214** is an instance of a web-based application that is hosted on an application server (e.g., application server **130**).

[0237] Local database **216** can include a database that can be used by local application **214**. For example, local database **216** can include, but is not limited to, MySQL, PostgreSQL, ORACLE, etc. In some embodiments, local database **216** includes a user database and an application database. The user database is described in more detail below with reference to FIG. **4**. The application database is described in more detail below with reference to FIG. **3**.

[0238] Stack **218** can include a number of software packages that enable a web-based application to run on a client. In some embodiments, the software packages can include, but are not limited to, a web server (e.g., Apache HTTP Server), a database (e.g., MySQL, PostgreSQL, ORACLE), application servers (e.g., Apache Tomcat), scripting languages (e.g., Python, Ruby, Ruby on Rails, etc.), and libraries. In some embodiments, stack **218** includes open source software (OSS) packages. In some embodiments, the OSS packages include Apache HTTP server, MySQL, PostgreSQL, Apache Tomcat, Python, and libraries. Stack **218** is described in more detail below with reference to FIG. **12**.

[0239] Synchronization application **220** can synchronize a web-based application that is running on a client (e.g., clients **210**-B) with a corresponding web-based application running on an application server (e.g., application server **130**). Synchronization application **220** is described in more detail below with reference to FIGS. **6** and **23**-**28**.

[0240] Traffic management module **222** can manage network traffic between the client and other computer systems. For example, if a user using client **210**-A enters a universal resource locator (URL) into browser **212**, traffic management module determines an Internet protocol (IP) address for the URL. In some embodiments, traffic management module **222** first looks at a hosts file for client **210**-A. The hosts file can map URLs to IP addresses. In some embodiments, for web-based application that are installed on client **210**-A, the hosts file can include an entry that associates an IP address for client **210**-A (e.g., 127.0.0.1) with the URL. If an entry for the URL exists in the hosts file, traffic management module returns the IP address associated with the URL. If an entry for the URL does not exist in the hosts file, traffic management module **222** can query a dynamic naming service (DNS) server to determine an IP address for the URL. The DNS server may be on client **210**-A or may be on a remote DNS server. Note that if a network connection to a remote DNS server does not exist, the client computer system may resort to using the local hosts file and/or the local DNS server to resolve IP addresses. If an entry for the URL is found in a DNS server, traffic management module **222** returns the IP address associated with the URL. If an IP address associated with the URL is not found, an error message can be returned.

[0241] License authentication module **224** can be used to verify whether a given user or a given client can use a web-based application loaded on the given client. Copy protection and data security module **226** can protect data that is located in local database **216** and/or local application **214** from being accessed by unauthorized users. For example, copy protection and data security module **226** can protect data by using access control mechanisms to restrict access to data. Alternatively, copy protection and data security module **226** can

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 188

protect data by encrypting the data. Copy protection and data security module **226** can also work with license authentication module **224** to prevent users that have expired licenses from accessing data stored in local database **216**.

[0242] As illustrated in FIG. **2**, client **210**-B is operating in an AOP mode whereas client **210**-A is operating in a networked mode. In a networked mode, a user on client **210**-A can access a web-based application on application server **130** through network **120**. In contrast, a user on client **210**-B does not have a connection to application server **130**, and thus cannot access functionality for a web-based application that requires a network connection to application server **130**. However, since client **210**-B has the AOP framework installed, client **210**-B can operate in an AOP mode where the full functionality of web-based application (including functionality that normally requires a network connection to application server **130**) is available to the user. In some embodiments, a local web server **228** runs web-based application locally. In some embodiments, local web server **228** can be part of stack **218**. Note that client **210**-A can also include local web server **228**. If client **210**-A loses a connection to network **120**, client **210**-A can load the AOP framework and use a local web server to run the web-based application locally until the network connection is restored.

[0243] Application server **130** can include one or more of: lightweight directory access protocol (LDAP) gateway **242**, web server engine **246**, auxiliary services **250**, synchronization, access and query engine **260**, applications database **262**, user database **270**, and ecommerce services **280**. LDAP gateway **242** can provide directory services to clients through network **120**. Note that LDAP gateway **242** is optional in some embodiments.

[0244] Web server engine **246** can respond to requests for static web pages, dynamically generated web pages, and web-based applications. These requests can come from clients (e.g., **210**-A and **210**-B) or from other application servers. In some embodiments, web server engine **246** is an open source web server (e.g., Apache HTTP server). Web server engine **246** can access LDAP gateway **242**, auxiliary services **250**, user database **270**, synchronization, access, and query engine **260**, and e-commerce services **280**.

[0245] Auxiliary services **250** can include, but are not limited to: famfamfam (icon images), phpMyAdmin (php web-based MySQL database management interface), phpPGAdmin (php web-based PostgSQL database management interface), WebSVN (php web-based Subversion interface), ImageMagick (image manipulation library), ZendFramework (php utility framework), IconCube Loaders (encrypted-php decryption library), libpng (PNG manipulation library), libjpeg (JPEG manipulation library), Neon (WebDAV client library), mcrypt (encryption library), and FreeType (font utilities library).

[0246] Synchronization, access, and query engine **260** can synchronize data and files between a web-based application hosted on application server **130** and a corresponding web-based application hosted on a client (e.g., clients **210**-A and **210**-B). Synchronization, access, and query engine **260** can include rules for conflict management and techniques for handling automatically incrementing record identifiers in a database for web-based applications. Synchronization, access, and query engine **260** can access applications database **262**, which can include information about web-based applications available on the application server and can include data for the web-based applications. Synchronization, access, and query engine **260** is described in more detail below with reference to FIGS. **5** and **23**-**28** below. Applications database **262** is described in more detail below with reference to FIG. **3** below.

[0247] User database **270** can include information about users. In some embodiments, user database **270** can be used to track users that are allowed to access web-based applications on application server **130**. User database **270** is described in more detail below with reference to FIG. **4**.

[0248] E-commerce services **280** can provide payment and order fulfillment services. In some embodiments, e-commerce services **280** includes process payment module **282**, authenticate license module **284**, and serve application module **286**. Process payment module **282** can process payments for goods and services. For example, process payment module can authorize payments by credit card, debit cards, electronic funds transfers, or other payment mechanisms (e.g., credits, prepaid tokens, etc.). Authenticate license module **284** can verify that a user has a valid license for a specified service and/or web-based application. E-commerce services **280** can access user database **270** to determine license information and/or payment information. Serve application module **286** can serve applications to a user after a product or service has been purchased or after a license has been verified.

[0249] FIG. **3** presents a block diagram of application database **300**, according to embodiments of the present invention. Application database **300** includes records **302** to **306** for application **1** to application M, respectively. In some embodiments, applications **1** to application M are web-based applications hosted on an application server. Note that in general there can be any number of applications stored in applications database **300**. Each application can have a number of records associated with the application.

[0250] Record **302** for application **1** can include information about an application hosted on an application server. For example, record **302** can include, but is not limited to, application name **330**, application title **332**, application host **334**, application user **336**, and application password **338**. Application name **330** can be a name for the application. Application title **332** can be a title for the application. Application host **334** can be a URL and/or an IP address that is associated with the application on the application server. Application user **336** can be the user or group of users who can manage the application. Application password **338** can be a password for accessing the management features for the application on the application server. Records **304** to **306** for application **2** to application M, respectively, are similar to record **302** for application **1**.

[0251] Record **1** **312** for application **2** can include metadata and content for application **2**. For example, record **1** **312** can include, but is not limited to, record metadata **350**, record log **352**, and record content **1** **354**-**1** to record content N **354**-N. Record metadata **350** can include information about a given record. For example, the metadata can include, but is not limited to, the date and time the record was created, the user that created the record, and/or the IP address of the user who created the record. Record log **352** can include a log for record **312**. For example, record log **352** can be used when synchronizing application **2** between the application server and a client computer system. Record content **1** **354**-**1** to record content N **354**-N can include content for record **1** **312**. In some embodiments, record **1** **312** to record N **314** can be in the same table within application database **300**. In other embodiments, record **1** **312** to record N **316** are in different tables

within application database **300** or in other databases linked to application database **300**. The other records **308-310** and **314-318** are similar to record **312**.

[0252] Application index **360** is an index to the applications available in application database **300** (e.g., applications **1**, **2**, . . . M). For example, applications index can include exemplary applications such as Etelos CRM (ECRM) **370**, Media Wiki **372**, phpBB3 **374**, Projects **376**, Sugar CRM **378**, and/or Wordpress **380**.

[0253] In some embodiments, each application hosted on an application server is stored in a separate database. These separate databases can be located on the application server or on remote database servers.

[0254] Note that the discussion above is one example of an application database. The information included in the application database can include more or fewer records than what are illustrated in FIG. **3**.

[0255] FIG. **4** presents a block diagram of user database **400**, according to embodiments of the present invention. User database **400** includes a number of user records **402-1** to **402**-N. User database **400** also includes map **470** that can map user IDs to user records. For example, as illustrated in FIG. **4**, map **470** maps user ID **472** to user record **402-2**.

[0256] User records **402-1** and **402**-N are similar to user record **402-2**. Thus, the description of user record **402-2** below applies to user records **402-1** and **402**-N. User record **402-2** includes, but is not limited to, user ID **410**, user metadata **412**, query/contact list **414**, user client device ID/type **416**, user preferences **418**, user authentication information **420**, user personal information **422**, and user enabled features **424**. In some embodiments, user ID **410** can be a unique identifier for a user within user database **400**. In other embodiments, user ID **410** can be a unique identifier for a user across a specified set of databases (e.g., all or a subset of the database) in the AOP framework. User metadata **412** can include metadata information about the user record. For example, the metadata information can include, but is not limited to, a date and time when the user record was created, a user who created the user record, the IP address of the user who created the user record, etc.

[0257] Query/contact list **414** can include queries that are used to retrieve contact records for a user (e.g., user-rule **1** **460-1** to user-rule **460**-N). The user rules can also be used to retrieve contact information, sales opportunities, filter database information, rules, tasks, appointments, and other contact-related information. For a given contact within query/contact list **414**, information related to the contact can be stored in the contact records for the contact. For example, the information can include contact information for the contact (e.g., name, phone number, fax number, address, email address, etc.), action items due to the contact, a last interaction with the contact, etc.

[0258] User client device ID/type **416** can store information about the type or IDs of computer devices that the user has used to access applications on the application server. For example, user client device ID/type **416** can indicate that a user used a laptop and a PDA to access applications on the application server. This information can then be used to generate a response that is substantially optimized for the computer device that the user is using to access the application.

[0259] User preferences **418** can include preferences for the user. These preferences can be used when generating responses for the user. For example, the preferences can include, font types and sizes, color schemes, etc. User authentication information **420** can be used to authenticate a user. For example, the authentication information can be a username/password combination for the user or can be a digital certificate for the user.

[0260] User personal information **422** includes, but is not limited to, name **430**, phone number **432** (e.g., home, work, mobile, pager, fax, etc.), email addresses **434**, office information **436** (e.g., company name, office address, phone number, fax number, etc.), and/or department **438**.

[0261] User enabled features **424** includes a list of features on the application server that have been enabled for the user. In some embodiments, user enabled features **424** are determined by the license (e.g., subscription, free, purchased, etc.) granted to the user. In some embodiments, user enabled features **424** are determined by the features that were purchased by the user. In some embodiments, user enabled features **424** are determined by the web-based applications that are available on an application server. Exemplary user enabled features **424** can include, but are not limited to, Application **1** **440**, Application N **442**, EDE **444**, Devkit **446**, Schedule events **448**, user management **450**, AOP framework **452**, distribute **454**, and integrate **456**. User management **450** includes a list of rights and privileges for a user. For example, user management **462-1** can include, but is not limited to, administrative rights and access privileges for the user associated with user record **402-2**. In some embodiments, these rights and privileges can be determined based on the applications available for the user and/or user enabled features **424**. Note that user admin rights and access privileges are described in more detail below with reference to FIG. **9**.

[0262] Note that the information in the user records can be located within one or more tables of user database **400** and/or in other associated databases. Also note that the discussion above is one example of a user database. The information included in the user database can include more or fewer records than what are illustrated in FIG. **4**.

[0263] In some embodiments, user database **400** is a distributed database. In some embodiments, user databases **400** can be located on the application server or on remote database servers.

[0264] FIG. **5** presents a block diagram of synchronization engine **500** on an application server, according to embodiments of the present invention. Synchronization engine **500** includes a synchronization data module **502**, a conflict management module **504**, synchronization rules **506**, and synchronization operations **508**.

[0265] Synchronization data module **502** includes data used by synchronization engine **500** to synchronize a web-based application on a client computer system with a web-based application on the application server. As illustrated in FIG. **5**, synchronization data module **502** includes files **562**, application **510**, application table **512**, application column **514**, and primary key **516**.

[0266] Conflict management module **504** resolves conflicts between the application server and client computer systems. These conflicts can arise when client computer systems operate web-based applications while not connected to the application server. For example, if a web-based application uses an automatically incrementing identifier for database records, client computer systems that are not connected to the application server can unknowingly use the same identifiers as the application server for different data resulting in data conflicts. Thus, in some embodiments, a record increment module **520** is provided to resolve conflicts for web-based application that

use automatically incrementing identifiers for database records. Conflict management module **504** and record increment module **520** are described in more detail below with reference to FIGS. **25-28** below.

[0267] Synchronization rules **506** are rules used by synchronization engine **500** to determine how to synchronize records between a client computer system and an application server. These rules can include developer specified rules **522** and default rules **524**.

[0268] Synchronization operations module **508** includes, but is not limited to, create accounts module **540**, create databases module **542**, create account directories module **544**, install EAS module **546**, setup scheduler module **548**, and connect to EMS client **550**. Create accounts module **540** can be used to create accounts for web-based applications. Create databases module **542** can be used to create the databases for web-based applications and the AOP framework. Create account directories **544** can be used to create the directory structure of web-based applications. Install EAS module **546** can be used to install an application server that can be used to serve web-based applications to clients. In some embodiments, the application server module is the Etelos Application Server (EAS) module. Setup schedule module **548** can setup a synchronization schedule between the application server and client computer systems.

[0269] Connect to EMS client module **550** can be used to connect an application server with the EMS module on a client computer system. Connect to EMS client module **550** can send data to an EMS module on the client computer system to synchronize the web-based application between the application server and the client computer system. The data can include, but is not limited to, setup data **552**, schema **554**, files **556**, rules **558**, and/or data **560**.

[0270] FIG. **6** presents a block diagram of synchronization engine **600** on a client computer system, according to embodiments of the present invention. The modules in synchronization engine **600** perform similar functions as the modules in synchronization engine **500**. For example, synchronization data module **602** generally corresponds to synchronization data module **502** and synchronization operations **608** generally corresponds to synchronization operations **508**. As a result, the discussion above for synchronization engine **600** is not repeated.

[0271] In some embodiments, the file structures for a web-based application in the application server and the client computer system are similar.

[0272] FIG. **7** presents a block diagram of application server **700**, according to embodiments of the present invention. The application server **700** generally includes one or more processing units (CPU's) **702**, one or more network or other communications interfaces **704**, memory **710**, and one or more communication buses **708** for interconnecting these components. The communication buses **708** may include circuitry (sometimes called a chipset) that interconnects and controls communications between system components. The application server **700** may optionally include a display **706** and one or more input devices **705** (e.g., keyboard, mouse, trackpoint, etc.). Memory **710** includes high-speed random access memory, such as DRAM, SRAM, DDR RAM or other random access solid state memory devices; and may include non-volatile memory, such as one or more magnetic disk storage devices, optical disk storage devices, flash memory devices, or other non-volatile solid state storage devices. Memory **710** may optionally include one or more storage devices remotely located from the CPU(s) **702**. Memory **710**, or alternately the non-volatile memory device(s) within memory **710**, comprises a computer readable storage medium. In some embodiments, memory **710** stores the following programs, modules and data structures, or a subset thereof: operating system **711**, network communication module **712**, LDAP gateway module **714** (optional), synchronization, access and query module **716**, user database **728**, e-commerce services module **730**, web server engine module **738**, application database management module **744**, user database management module **754**, application database **764**, and/or auxiliary services modules **766**.

[0273] Operating system **711** includes procedures for handling various basic system services and for performing hardware dependent tasks. Network communication module **712** can be used for connecting the application server **700** to other computers via the one or more communication network interfaces **704** (wired or wireless) and one or more communication networks, such as the Internet, other wide area networks, local area networks, metropolitan area networks, and so on. LDAP gateway module **714** can provides directory services for application server **700** and other computer systems. In some embodiments, LDAP gateway module **714** is optional.

[0274] Synchronization, access, and query module **716** can provides synchronization procedures to synchronize a web-based application on a client computer system with a web-based application on an application server. Synchronization, access, and query module **716** includes one or more of synchronization module **718**, synchronization files module **719**, access database module **720**, query database module **722**, and/or conflict management module **724**. Synchronization module **718** can perform synchronization operations between a client computer system and application server **700**. Synchronization files module **719** synchronizes files between the client computer system and application server **700**. Access database module **720** performs read (e.g., select operations) and write operations (e.g., insert, update, and delete operations) on databases. Query database module **722** performs queries in the databases and returns results of the query. Conflict management module **724** resolves conflicts between application server **700** and client computer systems. Conflict management module **724** can include record increment module **726**, which can handle conflicts that arise from automatically incrementing identifiers for database records.

[0275] User database **728** includes user information as described above with reference to FIGS. **2** and **4**. Application database **764** includes application data as described above with reference to FIGS. **2** and **3**.

[0276] E-commerce services module **730** can provide electronic commerce services. E-commerce services module **730** includes one or more of process payment module **732**, authenticate license module **734**, serve application module **736**. E-commerce services module **730** is described in more detail above with reference to FIG. **2**.

[0277] Web server engine module **738** can serve web pages to client computer system or other application servers. In some embodiments, web server engine module **738** includes web development environment module **740**, which provides software and tools to build and serve web-based applications. In some embodiments, web development environment module **740** is LAMP module **741**, which includes the LINUX operating system, Apache HTTP server, the MySQL database management system, and the PHP scripting language.

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 191

[0278] The components of LAMP module **741** can be substituted for other compatible technologies. In general, LAMP module **741** includes an operating system, a web server, a database, and support for a scripting language. For example, LAMP module **741** can include WAMP (Windows, Apache HTTP, MySQL, PHP) and/or LAPP (Linux, Apache HTTP, PostgreSQL, PHP). LAMP module **741** can also include Apache Tomcat. In some embodiments, web server engine module **734** includes a Python module **742** that supports the Python scripting language. Optionally, Python module **742** can also support Ruby, Ruby on rails, and/or any other languages.

[0279] Application database management module **744** provides an interface to manage and access an applications database for web-based applications that are hosted on application server **700**. Application database management module **744** can include an add/delete application module **746**, a synchronize application module **748**, and an application access module **750**. Add/delete application module **746** allows for the addition or deletion of web-based application records in application database **764** on application server **700**. Synchronize application module **748** synchronizes applications database **764** with an application database on a client computer system. Application access module **750** determines whether a user is allowed to access a given application within application server **700**.

[0280] User database management module **754** provides an interface to manage and access a user database for web-based applications that are hosted on application server **700**. Users database management module **754** can include an add/delete user module **756**, an edit user information module **758**, a user authentication module **760**, and/or a user permissions module **762**. Add/delete user module **756** allows for the addition or deletion of users in user database **728** on application server **700**. Edit user information module **758** allows for the editing of user information in user database **728**. User authentication module **760** authenticates users by checking user credentials stored in user database **728** (e.g., by checking a username and password for a user, or a digital certificate). User permissions module **762** determines whether a user is allowed to access specified resources and/or applications within application server **700**.

[0281] Auxiliary services module(s) **766** includes, but is not limited to: famfamfam (icon images), phpMyAdmin (php web-based MySQL database management interface), phpPGAdmin (php web-based PostgSQL database management interface), WebSVN (php web-based Subversion interface), ImageMagick (image manipulation library), ZendFramework (php utility framework), IconCube Loaders (encrypted-php decryption library), libpng (PNG manipulation library), libjpeg (JPEG manipulation library), Neon (WebDAV client library), mcrypt (encryption library), and FreeType (font utilities library).

[0282] Each of the above identified elements may be stored in one or more of the previously mentioned memory devices, and corresponds to a set of instructions for performing a function described above. The above identified modules or programs (i.e., sets of instructions) need not be implemented as separate software programs, procedures or modules, and thus various subsets of these modules may be combined or otherwise re-arranged in various embodiments. In some embodiments, memory **710** may store a subset of the modules and data structures identified above. Furthermore, memory **710** may store additional modules and data structures not described above.

[0283] Although FIG. **7** shows an "application server," FIG. **7** is intended more as functional description of the various features that may be present in a set of servers than as a structural schematic of the embodiments described herein. In practice, and as recognized by those of ordinary skill in the art, items shown separately could be combined and some items could be separated. For example, some items shown separately in FIG. **7** could be implemented on single servers and single items could be implemented by one or more servers. The actual number of servers used to implement an application server and how features are allocated among them will vary from one implementation to another, and may depend in part on the amount of data traffic that the system must handle during peak usage periods as well as during average usage periods.

[0284] FIG. **8** presents a block diagram of client **800**, according to embodiments of the present invention. The client **800** generally includes one or more processing units (CPU's) **802**, one or more network or other communications interfaces **804**, memory **810**, and one or more communication buses **808** for interconnecting these components. The communication buses **808** may include circuitry (sometimes called a chipset) that interconnects and controls communications between system components. The client **800** includes a display **806** and one or more input devices **805** (e.g., keyboard, mouse, trackpoint, etc.). Memory **810** includes high-speed random access memory, such as DRAM, SRAM, DDR RAM or other random access solid state memory devices; and may include non-volatile memory, such as one or more magnetic disk storage devices, optical disk storage devices, flash memory devices, or other non-volatile solid state storage devices. Memory **810** may optionally include one or more storage devices remotely located from the CPU(s) **802**. Memory **810**, or alternately the non-volatile memory device(s) within memory **810**, comprises a computer readable storage medium. In some embodiments, memory **810** stores the following programs, modules and data structures, or a subset thereof: operating system **811**, network communication module **812**, DNS support module **813**, receive and process user input module **814**, display module **815**, browser engine module **816**, synchronization, access and query module **818**, e-commerce client module **826**, local web server engine module **834**, local application database module **848**, local user database management module **858**, application database **868**, and/or user database **870**.

[0285] Operating system **811** includes procedures for handling various basic system services and for performing hardware dependent tasks. Network communication module **812** can be used for connecting the client **800** to other computers via the one or more communication network interfaces **804** (wired or wireless) and one or more communication networks, such as the Internet, other wide area networks, local area networks, metropolitan area networks, and so on. DNS support module **813** provides DNS services for client **800**. Receive and process user input module **814** receives and processes user inputs received from input devices **805**. Display module **815** displays a user interfaces for applications running on client **800**. Browser engine module **816** can include any application with a rendering engine that can access data and/or services on a local or a remote computer system and render the results so that a user can view the data

and/or interact with the services. In some embodiments, browser engine module **816** is a web browser.

[0286] Synchronization, access, and query module **818** can provides synchronization procedures to synchronize a web-based application on a client computer system with a web-based application on an application server. Synchronization, access, and query module **818** includes one or more of synchronize files module **819**, synchronization module **820**, local copy of database module **822**, and edit local database module **824**. Synchronization module **820** can perform synchronization operations between client **800** and an application server. Local copy of database module **822** makes a local copy of databases located on an application server. Edit local database module **824** provides an interface to edit the local copy of the databases.

[0287] E-commerce client module **826** can provide electronic commerce services that enable use of web-based applications on client **800**. E-commerce client module **826** includes one or more of download and enable application module **828**, authenticate license module **830**, and/or copy protection and data security module **832**. Download and enable application module downloads and enables web-based applications from an application server. Authenticate license module **830** determines whether the license for a web-based application is valid. If so, a user is allowed to access the web-based application. Otherwise, a user is prevented from accessing the web-based application. Copy protection and data security module **832** can protect data that is located in local databases (e.g., an application database or a user database) from being accessed by unauthorized users. Copy protection and data security module **832** can work with authenticate license module **830** to prevent users that have expired licenses from accessing data stored in local databases.

[0288] Local web server engine module **834** can serve web pages to client **800** or to other computer systems. In some embodiments, local web server engine module **834** includes web development environment module **836**, framework language module **840**, AOP framework module **844**, and/or traffic management module **846**. Web development environment module **836** provides software and tools to build and serve web-based applications. In some embodiments, web development environment module **840** includes a number of open source software (OSS) packages. Thus, web development environment module is sometimes referred to as a software stack. In some embodiments, web development environment module **836** includes LAMP module **838**, which includes the LINUX operating system, Apache HTTP server, MySQL database management system, and support for the PHP scripting language and Apache Tomcat. The components of LAMP module **838** can be substituted for other compatible technologies (e.g., WAMP, LAPP, etc.). In general, LAMP module **838** includes an operating system, a web server, a database, and support for a scripting language. Framework language module **840** provides support for one or more programming languages used to implement the AOP framework. In some embodiments, framework language module **840** includes python module **842** which supports the Python, Ruby, and/or Ruby on Rails scripting language. AOP framework module **844** provides data structures and procedures for running web-based applications on a client computer system (e.g., client **800**). AOP framework module **844** is described in more detail below with reference to FIGS. **10-28**. Traffic management module **846** manages network traffic between client **800** and other computer systems. Traffic management module **846** is described in more detail above with reference to FIG. **2**.

[0289] Local application database management module **848** provides an interface to manage and access applications database **868** for web-based applications that are hosted on client **800**. Local application database management module **848** can include an add/delete record module **850**, an application database access module **852**, and/or an application database query module **854**. Add/delete record module **850** allows for the addition or deletion of web-based application records in application database **868**. Application database access module **852** performs reads and writes into application database **868**. For example, application database access module **852** can process requests for retrieving data, adding records, deleting records, and editing records. Application database query module **854** performs queries on and returns results from application database **868**.

[0290] Local user database management module **858** provides an interface to manage and access user database **870** for web-based applications that are hosted on client **800**. Local user database management module **858** can include an add/delete user module **860**, an edit user information module **862**, a user authentication module **864**, and/or a user permissions module **866**. Add/delete user module **860** allows for the addition or deletion of users in user database **870**. Edit user information module **862** allows for the editing of user information in users database **868**. User authentication module **864** authenticates users by checking user credentials stored in user database **870** (e.g., by checking a username and password for a user, or a digital certificate). User permissions module **866** determines whether a user is allowed to access specified resources and/or applications within client **800**.

[0291] Auxiliary services module(s) **864** includes, but is not limited to: famfamfam (icon images), phpMyAdmin (php web-based MySQL database management interface), phpPGAdmin (php web-based PostgSQL database management interface), WebSVN (php web-based Subversion interface), ImageMagick (image manipulation library), ZendFramework (php utility framework), IconCube Loaders (encrypted-php decryption library), libpng (PNG manipulation library), libjpeg (JPEG manipulation library), Neon (WebDAV client library), mcrypt (encryption library), and/or FreeType (font utilities library).

[0292] Application database **868** is described in more detail above with reference to FIGS. **2** and **3** above. User database **870** is described in more detail above with reference to FIGS. **2** and **4** above.

[0293] Each of the above identified elements may be stored in one or more of the previously mentioned memory devices, and corresponds to a set of instructions for performing a function described above. The above identified modules or programs (i.e., sets of instructions) need not be implemented as separate software programs, procedures or modules, and thus various subsets of these modules may be combined or otherwise re-arranged in various embodiments. In some embodiments, memory **810** may store a subset of the modules and data structures identified above. Furthermore, memory **810** may store additional modules and data structures not described above.

AOP Framework

[0294] FIG. **10** presents a block diagram **1000** of exemplary AOP framework **100** which provides offline access to web-based application **1006**, according to embodiments of the

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 193

present invention. In some embodiments, client computer system **1001** is coupled to application server **1005** through network **120**. Network **120** can include, but is not limited to, a local area network (LAN), a wide area network (WAN), the Internet, a mobile network, and/or a wireless network. Client computer system **1001** includes operating system **1002**, browser **1003**, and AOP framework **1010**. As illustrated in FIG. **1**, AOP framework **1010** includes software stack **1004** and web-based application **1007**. Software stack **1004** may be comprised of open source software, commercial software, or a combination of both. Note that the details of AOP framework **1010** are described in more detail below with reference to FIGS. **10B-28** below. Application server **1005** includes web-based application **1006**.

[0295] Consider a user who wants to access web-based application **1006** located on application server **1005**. For example, the URL for the web-based application can be http://appl.com. The user can use browser engine **1003** on client computer system **1001** to access the URL http://appl.com. When client computer system **1001** has a network connection with application server **1005**, application server **1005** can activate web-based application **1006** to respond to the request from browser engine **1003**.

[0296] On a client computer system without AOP framework **1010**, the user can only access the full functionality of web-based application **1006** when there is a network connection between the client computer system and application server **1005**. However, since client computer system **1001** includes AOP framework **100**, the user can access the full or substantially the full functionality of web-based application **1006** when there is no network connection between client computer system **1001** and the application server **1005**. As illustrated in FIG. **1**, AOP framework **1010** includes web-based application **1007**, which is a local instance of web-based application **1006**. Software stack **1004** provides the software required to execute web-based application **1006** on client computer system **1001**. In some embodiments, Software stack **1004** is the same set of software used on application server **1005** to run web-based application **1006**.

[0297] When client computer system **1001** does not have a network connection with application server **1005**, AOP framework **1010** can allow the user to continue having access to the functionality of web-based application **1006** by activating web-based application **1007** on client computer system **1001**. Note that the loss of a network connection with application server can result from the user manually disconnecting the network connection (e.g., through software or hardware) or can be a result of external factors (e.g., power outages, network outages, etc.). In some embodiments, AOP framework **1010** automatically activates web-based application **1007** when the network connection to application server **1005** no longer exists. In other embodiments, after the network connection to application server **1005** no longer exists, AOP framework **1010** waits for the user to indicate that web-based application **1007** should be activated.

[0298] FIG. **11** illustrates an exemplary process **1100** of using AOP framework **1010** to run web-based application **1006** while offline, according to embodiments of the present invention. Note that FIG. **11** corresponds to the block diagram in FIG. **10**. The process begins when client computer system **1001** receives an input from a user (**1102**). Client computer system **1001** starts a browser application (**1104**). Client **1001** then receives a user request to visit a web page or an application (**1106**). The user request can include a URL. Optionally, client computer system **1001** then detects if a connection cannot be made to the web page or the application (**1108**). For example, the user may be offline and may not have an Internet connection. If the connection cannot be made, then a web server on client computer system **1001** responds to the request (**1110**). Client computer system **1001** can then perform one or more of the optional steps **1112-1118**. Client computer system **1001** can run the application locally using the client's operating system (**1112**). Client computer system **1001** can run the software in conjunction with the AOP framework (**1114**). Client computer system **1001** can receive a user request to navigate the browser to a web address (e.g., a URL). For example, the URL can be http://appl.com.aop. Note that the extension ".aop" can be replaced with any other extension. AOP framework **1010** can then respond to the request for the specified URL and activates web-based application **1007** to respond to the request. Client computer system **1001** can then detect when a connection can be made to a remote web page or a server corresponding to the local virtual instance (e.g., application server **1005**) (**1118**). When the network connection between client computer system **1001** and application server **1005** is reestablished, any changes made in web-based application **1007** on client computer system **1001** can be synchronized with web-based application **1006** on application server **1005**. In some embodiments, client computer system **1001** is synchronized with application server **1005** using Network **120**.

[0299] In some embodiments, the AOP framework modifies the operating system's network hosts file so that application domains ending with a specified domain suffix are treated as local domains served by a local web server. For example, if the specified domain suffix is ".aop", an application may have the universal resource locator (URL) "appl.com.aop". In some embodiments, the local web server is on the same client computer system as the client computer system running the AOP framework. For example, if the AOP framework is running on a laptop, the laptop may also include a local web server.

[0300] In some embodiments, when a user attempts to use a browser to access an application that has the specified domain suffix, the request is handled by a local web server and control is delegated to the application associated with the domain name on the client computer system.

[0301] In some embodiments, the AOP framework supports web application development languages including, but not limited to, PHP, Python, Ruby, and/or Ruby on Rails.

[0302] In some embodiments, the AOP framework determines all local changes made to applications on the client computer system and synchronizes these changes with the application server. In some embodiments, the synchronization is performed on a predetermined frequency. In other embodiments, the synchronization is performed on demand.

AOP Installation

[0303] The AOP framework enables web-based applications to run locally without requiring a network connection to an application server hosting the web-based application. As a result, some embodiments install software packages on a client computer system. These software packages can include, but are not limited to a web server (e.g., Apache HTTP Server), a database (e.g., MySQL, PostgreSQL), application servers (e.g., Apache Tomcat), scripting languages (e.g., Python), and libraries. In some embodiments, the software package includes Apache HTTP server, MySQL, Post-

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 194

greSQL, Apache Tomcat, Python, and libraries. In other embodiments, the software packages are included in the package for the AOP framework.

[0304] FIG. **12** presents a block diagram **1200** of an exemplary process of installing an instance of an AOP framework on a client computer system, according to embodiments of the present invention. The installation process includes downloading OSS packages (**1201**) and an AOP package (**1202**), installing the software packages (**1203**), installing the AOP package (**1204**), creating an account (**1205**), initializing an account (**1206**), connecting to the application server (**1207**), downloading application files from the application server (e.g., using RPC and SVN through Network **120**), and setting up an AOP application on the client computer system (**1209**).

[0305] FIG. **13** illustrates an exemplary process **1300** of installing an instance of an AOP framework on a client computer system, according to embodiments of the present invention. Note that FIG. **13** corresponds to the block diagram in FIG. **12**. The process begins when a user downloads software packages (**1301**). The software packages can be included in a single compressed file. The user also downloads the AOP framework package (**1302**). The AOP framework package can be included in a compressed file. In some embodiments, the AOP framework package includes an installation utility.

[0306] The user then installs the software packages on the client computer system (**1303**). If the software packages are already installed on the client computer system, the software packages are not reinstalled. In some embodiments, if the software packages are already installed on the client computer system, the versions of the software packages installed on the client computer system are determined. The software packages are either updated or not updated based on the determined versions. For example, if a minor software version update occurred for a software package installed on the client computer system, the minor software version update may not be applied. In some embodiments, if the software packages are already installed on the client computer system, the software packages may periodically check for updates and apply the updates according to a set of rules. For example, the set of rules can specify that certain software packages can be updated without user intervention whereas others require specified instructions from users and/or the developers of the web-based application. After the software packages are installed, the user starts the AOP package installation process (**1304**). The AOP package installation process installs the AOP framework. During the installation process, the user enters account information about web-based applications that the user desires to use (**1305**). For example, the account information can include one or more of universal resource locator (URL) for the web-based application, a username, and a password.

[0307] The AOP framework then initializes the account (**1306**). The account initialization operation is described in more detail with reference to FIGS. **14-15** below.

[0308] After the account is initialized, the client computer system connects to the application server so that the application server can authenticate the user and verify AOP permissions (**1307**). After the user has been verified, the application server transfers information required to complete the installation of the web-based application on the client computer system (**1308**). For example, the application server can transfer files and data for the web-based application to the client computer system. The client computer system then installs up the web-based application (**1309**). Step **1309** is described in more detail below with reference to FIGS. **16-17**.

AOP Initialize Account

[0309] FIG. **14** presents a block diagram **1400** of an exemplary process of initializing an AOP account on a client computer system, according to embodiments of the present invention. FIG. **14** includes AOP application **1401**, vhost file **1405**, AOP initialization file **1406**, database **1408-1409**, packages **1414**, account **1404**, EMS **1402**, software stack **1410**, operating system **1412**, host file **1407**, download **1408**, connect to server **1403**, and Network **120**.

[0310] FIG. **15** illustrates an exemplary process **1500** of initializing an AOP account on a client computer system, according to embodiments of the present invention. Note that FIG. **15** corresponds to the block diagram in FIG. **14**. The AOP framework is installed on the client computer system (**1501**). After the AOP framework has been installed on the client computer system, a database for an account management system is installed (**1502**). In some embodiments, the account management system is the Etelos Management System (EMS). The term EMS is used below to describe a generic account management system. The AOP framework connects to the application server and authenticates user permissions (**1503**). If the user is authenticated, an account is created in the account management system (**1504**).

[0311] The local web server instance within the AOP framework is configured to enable navigation to the web-based application that is installed on the client computer system (**1505**). In some embodiments, a virtual hosts (vhost) file is configured to map a specified URL to a location for the web-based application on client computer system. For example, the vhost file can map http://appl.com.aop to /web/appl. As noted above, the ".aop" suffix can be any suffix.

[0312] An AOP initialization file that includes additional configuration information is saved (**1506**). The specified URL is then associated with the client computer system (**1507**). In some embodiments, an entry is added to a hosts file on the client computer system. The entry can be used to redirect a request for the specified URL to the client computer system. In other embodiments, an entry is added to a dynamic naming service (DNS) server on the client computer system. The entry can be used to redirect a request for the specified URL to the client computer system.

[0313] Packages that are required by the web-based application are then downloaded from a package repository onto the client computer system (**1508**). These packages can include packages such as EAS, Zend, fam fam fam, phpMyAdmin, etc (**1509**). The packages are then installed (**1510**).

[0314] The account is initialized and the system will move onto application installation.

Web-Based Application Installation

[0315] FIG. **16** presents a block diagram **1600** of an exemplary process of installing a web-based application on a client computer system, according to embodiments of the present invention. FIG. **16** includes AOP framework **1601**, software stack **1613**, operating system **1614**, files **1615**, schema **1616**, data **1617**, rules **1618**, Network **120**, and connect to server **1620**. AOP Framework **1601** includes application **1602**, packages **1607**, account **1608**, EMS **1609**, and databases

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 195

1610-1611. Application **1602** includes vhost **1603**, web files **1604**, file transfer **1605** (e.g., SVN, rsync, etc.), and database **1606**.

[0316] FIG. **17** illustrates an exemplary process **1700** of installing a web-based application on a client computer system, according to embodiments of the present invention. Note that FIG. **17** corresponds to the block diagram in FIG. **16**. The client computer system downloads and installs the schema of the database for the web-based application from the application server (**1701**). The client computer system also downloads the files required by the web-based application (**1702**). For example, the files can be downloaded using Subversion (SVN), which is a version control system, and/or rsync. The client computer system then downloads synchronization rules and inserts the rules into the database for the account management system (**1703**). Next, the client computer system downloads application data as part of the initial synchronization process (**1704**). Step **1704** is described in more detail below with reference to FIGS. **18-19**.

Initial Data Synchronization

[0317] FIG. **18** presents a block diagram **1800** of an exemplary process of performing an initial data synchronization for a web-based application on a client computer system, according to embodiments of the present invention. FIG. **18** includes AOP framework **1801**, software stack **1813**, operating system **1814**, files **1817**, data **1818**, Network **120**, and connect to server **1816**. AOP Framework **1801** includes application **1802**, packages **1807**, account **1808**, EMS **1809**, synchronizer **1810**, and databases **1811-1812**. Application **1802** includes vhost **1803**, web files **1804**, file transfer **1805** (e.g., SVN, rsync, and other file transfer tools), and database **1806**.

[0318] FIG. **19** illustrates an exemplary process **1900** of performing an initial data synchronization for a web-based application on a client computer system, according to embodiments of the present invention. Note that FIG. **19** corresponds to the block diagram in FIG. **18**. The initial data sync can be a large and time consuming process because all data records for the web-based application on the application server are synchronized with the web-based application on the client computer system. The synchronization rules processing can be more simple because a new installation typically does not have conflicting data between the application server and the client computer system.

[0319] The process begins when a synchronization engine within the AOP framework on the client computer system starts an initial data synchronization process (**1901**). The synchronizer then retrieves initial data synchronization rules from the database (**1902**). The synchronizer sends a request to the application server to synchronize with the application server (**1903**). This request can include user authentication information. Once the user is authenticated, the client computer system synchronizes files for the web-based application using a file transfer utility (**1904**). For example, the file transfer utility can include Subversion (SNV), rsync, etc. The application server then sends data records to the client computer system (**1905**). In some embodiments, the data records include all data records for the web-based application.

Using an the AOP Framework

[0320] FIG. **20** presents a block diagram **2000** of an exemplary process of using the AOP framework on a client computer system, according to embodiments of the present invention. FIG. **20** includes AOP framework **2001**, software stack **2012**, operating system **2013**, Network **120**, connect to server **2015**, AOP launch interface **2016**, launch browser with application running locally **2017**. AOP Framework **2001** includes application **2002**, packages **2007**, account **2008**, EMS **2009**, and databases **2010-2011**. Application **2002** includes vhost **2003**, web files **2004**, file transfer **2005** (e.g., SVN, rsycn, and other file transfer tools), and database **2006**.

[0321] FIG. **21** illustrates an exemplary process **2100** of using the AOP framework on a client computer system, according to embodiments of the present invention. Note that FIG. **21** corresponds to the block diagram in FIG. **20**. The AOP framework allows a user to use web-based applications on a client computer system regardless of whether the client computer system is connected to an application server that hosts the web-based application. In order to use web-based applications on the client computer system without a connection to the application server, the AOP framework must be running. A user on a client computer system executes the AOP framework on the client computer system (**2101**). The AOP framework loads the software stack (**2102**) and the EMS (**2103**). The AOP framework then loads a user interface (**2104**) and web-based applications that are available on the client computer system (**2105**). In some embodiments, the user interface can include a desktop interface (**2109**). A user can then select a web-based application to use. This selection is detected and causes the browser to navigate to the URL associated with the selected application (**2106**). The AOP framework causes the client computer system to be directed to a web server running on the client computer system (**2107**). For example, an entry in a hosts file or an entry in a DNS server on the client computer system can be used to direct the client computer system to the web server running on the client computer system. The web server on the client computer system listens for connections and responds to a connection request by serving the requested web-based application (**2108**). In some embodiments, the web server includes Apache HTTP server (**2110**).

[0322] A user that is using a web-based application on the application server can later lose a network connection to the application server. In some embodiments, if the client computer system is running the AOP framework, the user can continue using the web-based application without the network connection to the application server. In these embodiments, the AOP framework can seamlessly continue providing the services required by the web-based application so that the user does not know that the network connection to the application server is lost. After the network connection to the application server is restored, any changes made on the client computer system and the application server can be synchronized with each other.

[0323] In some cases, a network connection may be slow or unreliable. Thus, in some embodiments, a user can use a web-based application from the client computer system regardless of whether a network connection to the application server exists or not. In these embodiments, the AOP framework periodically synchronizes changes made on the application server and the client computer system with each other. In doing so, a poor user experience that could result from a low-quality or intermittent network connection can be reduced or eliminated.

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 196

[0324] In some embodiments, a web browser is used to access the web-based application.

Client Computer System

[0325] When a user requests a web-based application by entering a URL into a browser, the client computer system determines how to access the web-based application. FIG. **22** presents a block diagram **2200** of an exemplary process for a client computer system determining how to access a web-based application, according to embodiments of the present invention. FIG. **22** includes browser **2201**, host file **2202**, web server **2203**, vhost file **2204**, application **2205**, and packages **2209**. Application **2005** includes vhost **2206**, web files **2207**, file transfer **2208** (e.g., SVN, rsycn, and other file transfer tools), and database **2209**.

[0326] FIG. **23** illustrates an exemplary process **2300** for a client computer system determining how to access a web-based application, according to embodiments of the present invention. FIG. **23** corresponds to the block diagram in FIG. **22**. The client computer system first determines an IP address associated with the URL (**2301**). In some embodiments, the URL includes a domain suffix ".aop". The client computer system first checks a local hosts file to determine whether the URL is included in an entry in the hosts file (**2302**). If the URL is not included in an entry in the hosts file, the client computer system checks one or more DNS servers to locate the IP address associated with the URL. If the URL is included in an entry in the hosts file, the client computer system retrieves the IP address for the URL (**2303**). The IP address for the web-based application is set to an address associated with the client computer system (e.g., 127.0.0.1) (**2310**). In some embodiments, the IP address associated with the URL for web-based applications is set to an address associated with the client computer system each time the AOP framework is started on the client computer system. In other embodiments, the IP address associated with the URL for web-based applications is set to an address associated with the client computer system when the web-based application is installed on the client computer system.

[0327] The web browser is then directed to the IP address associated with the client computer system on which the web server on the client computer system listens (**2304**). The local web server receives the request (**2305**) and compares the requested URL to a virtual host (vhost) configuration file (**2306**) to determine where the files for the web-based application are located on the client computer system (**2307**) (e.g., the local application web root). The web-based application and tools required to complete the request are loaded (**2308**). In some embodiments, the tools include the AOP framework, database, etc (**2311**). The web server then returns a web page with the results of the request (**2309**).

AOP Synchronization

[0328] FIG. **24** presents a block diagram **2400** of an exemplary process for synchronizing a web-based application **2403** on a client computer system **2401** with a web-based application **2406** on an application server **2402**, according to embodiments of the present invention. As illustrated in FIG. **24**, client computer system **2401** includes web-based application **2403**, database **2404**, and changes log **2405**. Application server **2402** includes web-based application **2406**, database **2407**, and conflicts log **2408**. Web-based application **2403** corresponds to web-based application **2406**. Similarly, database **2404** corresponds to database **2407**.

[0329] FIG. **25** illustrates an exemplary process **2500** for synchronizing a web-based application on a client computer system with a web-based application on an application server, according to embodiments of the present invention. FIG. **25** corresponds to the block diagram in FIG. **24**. A synchronization process (or a synchronizer module) on the client computer system detects changes in a database **2404** for the client computer system **2401** (**2501**). In some embodiments, the synchronization process periodically polls database **2404** to determine whether changes have been made (**2508**). In other embodiments, the synchronization process listens for notification messages which are sent when database **2404** has been changed (**2508**).

[0330] The synchronization process then collects all the changes from database **2404** into a changes log **2405** (**2502**). The changes are sent to application server **2402** (**2503**). A synchronization process (or synchronization module) on the application server applies the changes received from the client computer system **2401** to database **2407** (**2504**). In some embodiments, the connection between client computer system **2401** and application server **2402** is kept alive until the synchronization process is completed.

[0331] If conflicts exist between the changes made on client computer system **2401** and application server **2402**, the data on the server "wins" (**2505**). The conflicting data is resolved at application server **2402** and conflict resolution data is collected in a conflict resolution log **2408**.

[0332] The synchronization process on application server **2402** then sends the conflict resolution log **2408** to client computer system **2401** (**2506**). The conflict resolution log is applied to database **2407** on client computer system **2401** (**2507**). Both client computer system **2401** and application server **2402** are synchronized at this point.

[0333] FIG. **26** presents a block diagram **2600** of an exemplary process for synchronizing a web-based application on a client computer system **2601** with a web-based application on an application server **2602**, according to embodiments of the present invention. FIG. **26** includes client **2601** and server **2602**. Client **2601** includes client in process **2606**, client applications database **2607**, client logs **2608**, and client out process **2609**. Application server **2602** includes server applications database **2603**, server logs **2604**, server out process **2605**, and server in process **2610**.

[0334] FIGS. **27**A-**27**F illustrate an exemplary process **2700** for synchronizing a web-based application on a client computer system **2601** with a web-based application on an application server **2602**, according to embodiments of the present invention. FIG. **27** corresponds to the block diagram in FIG. **26**. A user makes changes to the web-based application on application server **2602** (**2701**). For example, the user can make changes to a database for a web-based application by adding new records or modifying existing records in the database.

[0335] Database triggers watch for changes in database **2603** and note these changes in an internal log (**2702**). In some embodiments, if a rule type is "auto-increment," then a flag is added to the log (**2730**). A synchronization process on application server **2602** then pulls data from the internal log and sends the data to an account log (**2703**). The data in the account log is then sent to an outbound log (**2704**). The outbound log is used to speed up the synchronization process because the outbound log can become large and slow to

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 197

process. The synchronization process then waits for a user to initialize a synchronization operation with the application server (**2705**).

[0336] Once a synchronization operation is started, the data is sent to server out process **2605** for outbound processing (**2706**). In some embodiments, a transaction id called "sync block ID" is added to the data. The server out process performs several operations (**2707**), which can include, but are not limited to: applying filters, mapping data to users, sending user data to user tables, running group management rule checks, identifying administrative group memberships relationships for users, and/or mapping additional data to admin users based on a group admin of user A. The data is then sorted by users (**2708**). In some embodiments, the data is separated into separate "user out" tables, wherein each user has a "user out" table. In doing so, the speed of the synchronization can be optimized.

[0337] Client computer system **2601** receives the data and sends the data to client in process **2606** for processing (**2709**). Client in process **2606** performs a number of operations (**2710**), including, but not limited to, filtering Auto Increment Rules (flag), checking auto increment (AI) flagged column for conflict, marking data as "insert" if there are no conflicts, marking data in the sync log as "leave" (which are processed in the next session) if there are conflicts (see **2724**), setting an auto-increment counter to max+1, filtering data into inserts, updates and deletes, and/or sorting priorities (e.g., inserts before updates).

[0338] The synchronization process then inserts data that is marked as "insert" into client applications database **2607** for the web-based application (**2711**). The synchronization process then sends data that is marked as "update" or "delete" to a sync account log (**2712**). The updates and deletes in the sync account log are then processed by client applications database **2607** (**2713**). In some embodiments, when the synchronization process inserts data, the synchronization process disables triggers within a synchronization session to prevent doubling back into an infinite loop (**2713***a*). The web-based application on client computer system **2601** is now synchronized with the web-based application on application server **2602** (**2714**).

[0339] A user using web-based application on client computer system **2601** changes data in client applications database **2607** for the web-based application (**2715**). Database triggers watch for changes in client applications database **2607** and notes these changes in an internal log (**2716**). A synchronization process on client computer system **2601** then pulls data from the internal log and sends the data to an account log (**2717**). The data in the account log is then sent to an outbound log (**2718**). The outbound log is used to speed up the synchronization process because the outbound log can become large and slow to process. The synchronization process then waits for a user to initialize a synchronization operation with the application server (**2719**).

[0340] Once a synchronization operation is started, the data is sent to client out process **2609** for outbound processing (**2720**). In some embodiments, a transaction id called "sync block ID" is added to the data. Client out process **2609** performs several operations (**2721**), including, but not limited to, mapping data to sync account (user info is tied to sync acct) and/or pushing data to the sync account out log. Sync account out log is broadcast to application server **2601** (**2722**).

[0341] In some embodiments, every user has a table associated with the user (e.g., "in-table"), which is used for inbound processing. User in-tables (e.g., user A in-table, user B in-table) are pushed to server in process **2610** (**2723**). Server in process **2610** performs a number of operations (**2724**), including, but not limited to, comparing data—user out and user in—duplicate transactions, based on _eas_syncmap_id (added by trigger) and "sync_block_id" added by sync session, server wins—so server deletes conflicts, filtering inserts from updates and deletes, filtering auto increment flags, inserting records with no conflict, and/or inserting records with conflicts at the next unused record. Several examples of conflicting records include: _eas_sync_map_id=_eas_sync_map_id, Contact10=contact10, and Email=john@etelos vs. email=john.smith@etelos.com.

[0342] Data records that are marked as "insert" are inserted (**2725**). Data records that are marked as "update" or "delete" are sent to a sync account log (**2726**). The updates and deletes in the sync account log are then processed by server applications database **2603** (**2727**). In some embodiments, when the synchronization process inserts data, the synchronization process disables triggers this synchronization session to prevent doubling back into an infinite loop (**2732**). The web-based application on application server **2602** is now synchronized with the web-based application on client computer system **2601** (**2728**).

[0343] If the automatic incrementing rules detect a conflict between application server **2602** and client computer system **2601**, another pass is taken through the process to update conflicted records (**2729**). During this process, records that were previously left out because of conflicts are synchronized.

[0344] In some embodiments, the synchronization process can be used to synchronize between different types of database management systems (DMBSs). In some embodiments, the synchronization process can be used to synchronize between the same type of DBMSs. In some embodiments, the changes are synchronized directly with a DBMS associated with the web-based application. In all of these embodiments, database drivers are used to perform the synchronization between the DBMSs. The database drivers can handle the translations of data between different types of DBMSs as well as determining what changes occurred since the last synchronization process.

[0345] In some embodiments, the synchronization process synchronizes directly with the file structures associated with the web-based applications. In these embodiments, a file synchronization utility such as SVN or rsync can be used to synchronize the file structures.

Auto Increment Conflict Resolution

[0346] Some web-based applications use automatically incrementing identifiers in a database when a new record is added to the database. In these web-based applications, conflicts can arise if multiple instances of the web-based application (e.g., on the application server, on multiple client computer systems, etc.) are all inserting new data records into a database for that instance of the web-based application. Thus, some embodiments provide a technique to resolve conflicts for web-based application that use auto-increment identifiers.

[0347] FIG. **28** presents a block diagram **2800** of an exemplary process for resolving conflicts for web-based applications that use auto-incrementing identifiers, according to embodiments of the present invention. FIG. **28** illustrates the state of data on an application server and a client computer system at several points in time (e.g., before synchronization,

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 198

after a first pass of synchronization, after a second pass of synchronization, and after synchronization is complete).

[0348] FIG. **29** illustrates an exemplary process **2900** for resolving conflicts for web-based applications that use auto-incrementing identifiers, according to embodiments of the present invention. FIG. **29** corresponds to the block diagram in FIG. **28**. As illustrated in FIG. **29**, both the application server and the client computer system have added new records into their respective databases for the web-based application. The application server and the client computer system also have records that are already synchronized. Note that in FIG. **29**, the identifiers increase from left to right. Thus, the new records on the application server and the client computer system include a number of records that use the same identifiers, resulting in conflicts.

[0349] At the start of a synchronization operation, the data that is already synchronized on both the application server and the client computer system (e.g., the AOP instance) is identified (**2901**). The application server then determines that new records have been added to the application server since the last synchronization with the client computer system (**2902**). The client computer system determines that new records have been added to the client computer system since the last synchronization with the application server (**2903**).

[0350] The conflict resolution process begins when the application server sends its new records to the client computer system (**2904**). Since there is a conflict between the identifiers used in the application server and the client computer system, the conflict resolution process waits for a second pass before inserting the new records received from the application server. On the second pass, the new records from the application server are treated as an update.

[0351] The client computer system sends its new records to the application server (**2905**). The application server assigns new identifiers for these records and inserts these records into the database for the web-based application (**2906**). The application server can assign the new identifiers for these records by using the automatically incrementing identifiers in the database. The application server then generates an update log that includes the changes for these records (**2907**). The application server sends the update log to the client computer system which then processes updates from that result from these records and from the new records received from the application server at step **2804** (**2908**). The records in both databases are now synchronized (**2909**).

Users and Groups

[0352] FIG. **9** presents a block diagram illustrating exemplary group memberships for users for a given web-based application, according to embodiments of the present invention. As illustrated in FIG. **9**, user A is a member of group A. This membership can be managed through a memberships table. User B is a member of a group of administrators that are administrators of group A. This administrator membership can be managed through an admin table. In some embodiments, changes to user A's information are also added to a user out log table for user B. Thus, user B can see changes that user A has made. For example, a sales manager can view contacts for sales representatives that report to the sales manager.

[0353] In some embodiments, a group sync app group filter is used to indicate whether the web-based application supports group memberships. These rules can be setup in the account database for the web-based application. The filter can include information such as a data table (e.g., a contacts table), a membership table (e.g., the membership table), and a group table (e.g., groups). This can be a one time setup that a developer of a web-based application does to configure the web-based application to synchronize with the AOP framework. In some embodiments, the filter is setup when the developer ports the application to the AOP framework environment.

[0354] In some embodiments, the group filter sets flags in database triggers to send memberships data with a log entry so that the AOP framework knows what to use to filter in the server out process. In other words, the group filter is used to indicate which table includes user information, whether there is a membership table for linking users to groups, and the table that keeps track of groups.

Rules Setup

[0355] FIG. **36** presents a block diagram **3600** illustrating an exemplary user interface **3600** for creating synchronization rules, according to embodiments of the present invention. User interface **3600** includes one or more of: new synchronization rule interface **3602**, auto increment function **3622**, and/or create new sync rule function **3624**. New synchronization rule interface **3602** includes one or more of: application interface **3604**, mass create interface **3608**, save function **3626**, and/or close function **3628**.

[0356] Application interface **3604** includes one or more of database function **3610**, table function **3612**, column function **3618**, and/or primary key function **3620**. Database function **3610** allows a user to select available databases. Table function **3612** allows a user to select tables within the available databases. Column function **3618** allows a user to select columns within the tables. Primary key function **3620** allows a user to select primary keys for the tables. Using these functions, a user can specify synchronization rules on one or more of the database level, the table level, the column level, and the primary key level.

[0357] Mass create interface **3608** includes a mass create function which allows a user to automatically create sync rules for the AOP framework at every match for a specified level. For example, a user can use the mass create function to create sync rules for all database tables.

[0358] Auto increment function **3622** allows a user to specify which columns on a given table use automatically incrementing identifiers. Create new sync rule **3624** creates the new synchronization rule specified by the user in user interface **3600**. Save function **3626** saves the new rule whereas close function **3628** closes user interface **3600**.

[0359] FIG. **37** presents a block diagram **3700** illustrating an exemplary user interface **3700** for generating auto-incrementing identifiers, according to embodiments of the present invention. User interface **3700** includes auto increment definition interface **3702**, which includes one or more of application interface **3704**, relationships interface **3706**, and/or close function **3724**. Application interface **3704** includes one or more of application **1** **3708**, application **2** **3710**, and/or primary key **3712**. Primary key **3712** includes one or more of a table selection function, a column selection function, save function **3720**, and/or cancel function **3722**. Relationships interface **3706** includes one or more of create primary key reference function **3714**, contact ID function **3716**, groups description function **3718**, and an edit function.

[0360] FIG. **38** presents a block diagram **3800** illustrating an exemplary process for creating synchronization rules, according to embodiments of the present invention. FIG. **38**

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 199

includes one or more of account admin interface **3802**, select application and database **3804**, select table **3806**, select column **3808**, auto increment flag **3810**, rule create **3812**, application **3814**, and/or account **3822**. Application **3814** include one or more of database **3816**, trigger **3818**, and/or sync map ID **3820**. Account **3822** includes one or more of database **3824**, vmap sync rule **3826**, and AI flag **3828**.

[0361] FIG. **39** presents a flowchart of an exemplary process **3900** for creating synchronization rules, according to embodiments of the present invention. FIG. **39** corresponds to the block diagram in FIG. **38**. From the account admin interface (**3902**), a user can perform the following operations: selecting an application (which inherently selects the associated databases) (**3904**), selecting a table (**3906**), selecting a column (which can involve selecting a primary key in a table row for proper synchronization at this specific level) (**3908**), and/or optionally setting an auto increment flag (e.g., if relevant to the database configuration) (**3910**). After the user performs one or more of the previous options, the user can select a create rules function (**3912**), which calls functions to perform one or more of steps **3914-3922**. Alternatively, the user can refine the rules by using the account admin interface to specify more rules.

[0362] After selecting the create rules function, a rules generator creates a sync vmap rule in the account database (**3914**). The rules generator sets an auto increment flag if appropriate for rule type (e.g., if the database table uses an auto incrementing primary key column) (**3916**). The rules generator adds a column for a sync_map_id, which tracks sync transactions and ensures data integrity (**3918**). The rules generator creates a trigger in the database (**3920**). The synchronization rules are now setup at step **3922**. Note that a user can also select a "mass create" at one or more levels (e.g., all tables, specific tables, specific columns, etc.) in the process from now on that can automatically create sync rules for the AOP framework at every match at that level.

Foreign Key Mapping

[0363] Foreign key mappings specify the relationships between a parent table and a child table. These mappings are used during the synchronization and conflict resolution processes. FIG. **40** presents a block diagram **4000** of an exemplary foreign key mapping, according to embodiments of the present invention. The foreign key mapping includes parent **4002** and child **4020**. Parent **4002** includes one or more of identifier **4004**, application database identifier (e.g., App_DB_ID) **4006**, application table **4008**, application column **4010**, and auto increment flag **4012**. Child **4020** includes one or more of identifier **4022**, parent identifier **4024**, application database identifier (e.g., App_DB_ID) **4026**, application table **4028**, application column **4030**, and auto increment flag **4032**. Parent identifier **4024** for child **4020** corresponds to identifier **4004** for parent **4002**.

[0364] FIG. **41** presents a flowchart of an exemplary process **4100** for creating a foreign key mapping, according to embodiments of the present invention. In an account, rules for auto increment and relationships with foreign keys are created (**4102**). A parent is created (e.g., table Contact, ID, Name, etc.) (**4104**). A child is created (e.g., Groups ID, Contact.ID, Description, etc.) (**4106**). The parent-child relationship is updated (auto-inc=true) (**4108**). Auto increment any conflicts that may arise and updated parent-child table relationships in the data structure (**4110**).

Summary of AOP

[0365] FIG. **30** presents a flowchart of an exemplary process **3000** for providing access to a web-based application while offline, according to embodiments of the present invention. The process begins by providing on a computer system a local software stack configured to provide local web services for dynamic, web-based applications that are executed on the computer system when it is offline (**3002**). When the computer system is offline, the computer system executes a first dynamic, web-based application using the web services provided by the local software stack, such that functionality of the first dynamic, web-based application when the computer system is offline is substantially similar to functionality of the first dynamic, web-based application when the computer system is online (**3004**). In some embodiments, the first web-based application is a database-driven web-based application (**3006**). In some embodiments, the architecture of the first dynamic, web-based application is not modified to provide the functionality when the computer system is offline (**3008**).

[0366] In some embodiments, in response to detecting a network connection with an application server, the computer system synchronizes with the application server changes in information associated with the first dynamic, web-based application due to its offline execution (**3010**).

[0367] In some embodiments, a user of the computer system initiates execution of the first, dynamic web-based application when the computer system is offline by directing a web browser on the first computer system to a specified universal resource locator (URL) that is associated with the computer system instead of a remote application server (**3012**). The specified URL can be associated with the computer system through a modified hosts file on the first computer system (**3014**). Moreover, the specified URL can be associated with the computer system through a modified hosts file on the first computer system (**3016**). Furthermore, the specified URL can be associated with the first computer system through a dynamic naming server (DNS) record on the first computer system (**3018**).

[0368] In some embodiments, the local software stack comprises: a web server responsive to browser-issued commands, a database management system, and a dynamic scripting language (**3020**). In some embodiments, the web server is Apache web server, the database management system is mySQL, and the dynamic scripting language is at least one of PHP, Python, Perl, or Ruby (**3022**). In some embodiments, the first dynamic, web-based application communicates with the local software stack using TCP/IP (**3024**).

[0369] FIG. **31** presents a flowchart of an exemplary process **3100** for synchronizing a web-based application on a client computer system with a web-based application on an application server, according to embodiments of the present invention. For a web-based application that is not designed to be used while a first computer system on which the web-based application executes does not have a network connection to an application server (**3102**), changes are made to the web-based application while the first computer system does not have a network connection to the application server (**3104**). The changes made to the web-based application while the first computer system is disconnected from the application server are tracked (**3105**). When the network connection between

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 200

the first computer system and the application server is reestablished (**3106**), the changes are synchronized for the web-based application made on the first computer system with the web-based application on the application server (**3107**). The changes are synchronized for the web-based application on the application server with the web-based application on the first computer system (**3108**).

[0370] In some embodiments, the date and time when the changes to the web-based application were made are tracked (**3109**). In some embodiments, the changes are synchronized when a network connection between the first computer system and the application server is reestablished (**3110**). In some embodiments, the changes are synchronized between different types of database management systems (DBMS) (**3112**). In some embodiments, the changes are synchronized between the same types of database management system (DBMS) (**3114**). In some embodiments, the changes are synchronized directly with a database management system (DBMS) associated with the web-based application (**3116**). In some embodiments, the changes are synchronized directly with file structures associated with the web-based application (**3118**). In some embodiments, for a newly-installed instance of the web-based application on the first computer system, the method further comprises performing a complete synchronization of all records in the web-based application on the application server with the web-based application on the first computer system (**3120**). In some embodiments, the changes include one or more of: changes to data in the web-based application; changes to data structures in the web-based application; and/or changes to files for the web-based application (**3122**).

[0371] FIG. **32** presents a flowchart of an exemplary process **3200** for providing synchronizing a web-based application on a client computer system with a web-based application on an application server, according to embodiments of the present invention. For a web-based application that is being used on a first computer system while the first computer system is disconnected from an application server, in response to detecting a network connection to the application server (**3202**), changes made for the web-based application on the first computer system are synchronized with the web-based application on the application server (**3204**). The changes made for the web-based application on the application server are also synchronized with the web-based application on the first computer system (**3206**).

[0372] In some embodiments, for a newly-installed instance of the web-based application on the first computer system, a synchronization of records is performed in the web-based application on the application server with the web-based application on the first computer system (**3208**).

[0373] In some embodiments, the changes include one or more of: changes to data in the web-based application; changes to data structures in the web-based application; and/or changes to files for the web-based application (**3210**).

[0374] FIG. **33** presents a flowchart of an exemplary process **3300** for resolving conflicts between a web-based application on a client computer system and a web-based application on an application server, according to embodiments of the present invention. On a first computer system that is disconnected from an application server, a web-based application that is configured to interact over a network connection with the application server to provide specified functionality is used (**3302**). When the network connection is reestablished, changes made to the web-based application while the first computer system was disconnected from the application server are synchronized (**3304**). If a conflict between the first computer system and the application server exists, the conflicts are resolved so that both the first computer system and the application server are synchronized with each other (**3306**).

[0375] In some embodiments, a conflict exists if the first computer system includes a first set of records that are not included on the application server, and wherein resolving the conflict so that both the first computer system and the application server are synchronized with each other involves: sending the first set of records to the application server; receiving new identifiers for the first set of records, wherein the new identifiers are assigned by the application server when the first set of records are added to the application server; and updating all references to the old identifiers for the first set of records with the new identifiers for the first set of records (**3308**).

[0376] In some embodiments, a conflict exists if the application server includes a second set of records that are not included on the first computer system, and wherein resolving the conflict so that both the first computer system and the application server are synchronized with each other involves: receiving the second set of records from the application server; determining whether identifiers for the second set of records are already being used by the first computer system; and/or if the identifiers for the second set of records are not being used by the first computer system, inserting the second set of records (**3310**).

[0377] In some embodiments, if the identifiers for the second set of records are already being used by the first computer system, the following operations are performed: determining a third set of records that include identifiers that conflict with the identifiers for the second set of records; sending the third set of records to the application server; receiving new identifiers for the third set of records, wherein the new identifiers are assigned by the application server when the third set of records are added to the application server; updating all references to the old identifiers for the third set of records with the new identifiers for the third set of records; and inserting the second set of records (**3312**).

[0378] In some embodiments, if a conflict between the first computer system and the application server does not exist, records are updated on the first computer system so that the first computer system is synchronized with the application server (**3314**).

[0379] FIG. **34** presents a flowchart of an exemplary process **3400** for resolving conflicts between a web-based application on a client computer system and a web-based application on an application server that use automatically incrementing identifiers for database records, according to embodiments of the present invention. A first set of records are created with a corresponding first set of identifiers in a first database (**3402**). The first database is synchronized with a second database (**3404**). If the corresponding first set of identifiers already exists in the second database, a new set of identifiers is received for the first set of records from the second database, wherein the new set of identifiers is assigned to the first set of records when the first set of records is added to the second database, and all references to the corresponding first set of identifiers is updated for the first set of records with the new set of identifiers for the first set of records (**3406**).

[0380] In some embodiments, if the corresponding first set of identifiers does not exist in the second database, the first set of records is inserted into the second database using the corresponding first set of identifiers (**3408**).

[0381] In some embodiments, if the second database includes a second set of records that does not exist in the first database, the following operations are performed: receiving the second set of records from the second database; sending the first set of records to the second database; receiving new identifiers for the first set of records, wherein the new identifiers are assigned by the second database when the first set of records are added to the second database; updating all references to the old identifiers for the first set of records with the new identifiers for the first set of records; and inserting the second set of records (**3410**).

[0382] In some embodiments, the first database and the second database are used for a web-based application that requires a network connection with an application server to provide specified functionality (**3412**). In some embodiments, the web-based application was not designed to be used without a network connection to the application server (**3414**). In some embodiments, the first database is on a client computer system (**3416**). In some embodiments, the second database is on an application server (**3418**).

[0383] FIG. **35** presents a flowchart of an exemplary process **3500** for providing access to a web-based application while offline, according to embodiments of the present invention. A dynamic, web-based application configured to interact over a network connection with an application server to provide desired functionality is used (**3502**). The network connection is disconnected from the application server (**3504**). The web-based application continues to be used while still providing the desired functionality (**3506**).

[0384] In some embodiments, the dynamic, web-based application is a database-driven web-based application. In some embodiments, the architecture of the dynamic, web-based application is not modified to provide functionality when the computer system is offline. In some embodiments, in response to detecting a network connection with an application server, changes in information associated with the dynamic, web-based application due to its offline execution are synchronized with the application server. In some embodiments, a user of the computer system initiates execution of the dynamic, web-based application when the computer system is offline by directing a browser on the first computer system to a specified universal resource locator (URL) that is associated with the computer system instead of a remote application server.

[0385] Marketplace and Licensing

[0386] FIG. **44** is a high-level block diagram **4400** of a software marketplace, illustrating deployment of applications to a user account. A marketplace server **4410** makes available for distribution a plurality of software applications **4412**, **4414**, **4416**. A licensing engine **4420** associates licensing terms with the software applications. In some embodiments, the licensing engine **4420** is part of the marketplace **4410**. In some embodiments, the marketplace server is executed by a marketplace module **6122** (FIG. **61**).

[0387] One or more software applications (e.g., application **4412** and application **4414**) are deployed to an account **4432** hosted at a hosting infrastructure server **4430**. An account is a data structure associated with a user, maintained on a server (e.g., the hosting infrastructure server **4430**), where the account data structure includes information regarding which software applications are licensed to the user. In some embodiments the account may include license files, keys, etc. associated with one or more software applications. In some embodiments, applications licensed by a user are configurable to cross-synchronize, i.e. to synchronize data between the applications associated with the user. In some embodiments, the user associated with a user account could be an individual or an organization.

[0388] In some embodiments, the hosting infrastructure server is executed by a hosting module **6154** (FIG. **61**). In some embodiments, the deployment is across a network **4402**, e.g., where the hosting infrastructure server is located separately from the marketplace server **4410**. Deployment means the software application is associated with the user account, such that the user associated with the user account is permitted to access the software application. In some embodiments, one or more files relating to the software application are provided to the user account as part of the deployment.

[0389] In some embodiments, the hosting infrastructure server **4430** and the marketplace server **4410** are logical servers on one physical system, for example application sever **6100** (FIG. **61**). The deployment is performed in accordance with the licensing terms, provided by licensing engine **4420**, in some embodiments executed by licensing module **6126** (FIG. **61**). In some embodiments, the licensing engine controls the deployment of and access to software applications, and enforces license terms provided by the software vendor. In some embodiments, the licensing module provides the software code or executables to operate the licensing engine. The hosting infrastructure server **4430** includes an account management system (EMS) **4438** for managing the accounts **4432**, executed by user accounts module **6156** (FIG. **61**). A user account allows a user (person licensed to use software applications provided by the marketplace) to authenticate to system services. The user account also provides a user with the opportunity to be authorized to access deployed software applications. Users may need to identify themselves for the purposes of accounting, security, logging and resource management. In some embodiments, a user identifies him or herself using an account identifier and a username, and in some embodiments the user also has a password.

[0390] A software stack **4440** and operating system **4442** provide the software architecture to support the account framework and application execution. In some embodiments, the network **4402** is the Internet or an intranet or local connection between the marketplace **4410** and server **4340**.

[0391] One or more client systems **4450**, **4452**, **4454** access the hosting infrastructure server **4430** to execute the applications. In some embodiments, the client systems correspond to application client **6200**. In some embodiments, one or more of applications **4434**, **4436** are executed at the server **4430**, and results are displayed to the client through a client application or browser, executed by client/browser display module **6215**. In some embodiments, applications are run in a local mode (e.g., AOP mode as described earlier) where a client hosts and executes applications **4462** and **4464** associated with an account **4460**, corresponding to applications **4434** and **4436** hosted at hosting infrastructure server **4430**. An account management system (EMS) **4470** manages the accounts at the client, in some embodiments executed by user account access module **6272**. A client-side software stack **4472** and operating system **4474** provide the software architecture to support the account framework and application execution on the client.

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 202

[0392] FIG. **45** illustrates a block diagram **4500** of a software marketplace illustrating management of licenses in a multi-tenancy environment. A multi-tenancy environment is one which allows multiple customers or users to access a shared data model. In some embodiments, the software marketplace includes a licensing engine **4510**, executed by licensing module **6126**, to control licensing for hosted infrastructure servers (e.g., **4520**, **4530**). A hosted infrastructure server is one that hosts user accounts, and in some embodiments software applications associated with those user accounts. In some embodiments, the hosted infrastructure server may exist as a logical server on the same physical hardware as the marketplace server. In some embodiments, the hosted infrastructure server may be physically separate from the marketplace server. A multi-tenant license management engine **4512** generates and manages licenses (e.g., **4514**, **4516**) for deployment and execution of software applications on hosted infrastructure servers. The licenses are generated in response to a selection by a software vendor from options provided by the marketplace application. The license terms are associated with a software application. Software applications are deployed to the hosting infrastructure servers in accordance with the licenses (**4514**, **4516**).

[0393] In some embodiments, hosted infrastructure servers **4520**, **4530** are distributed tenants in a multi-tenant licensing system. In a distributed tenant architecture, each user runs its applications in a distributed-computing environment. In some embodiments, the hosted infrastructure servers **4520**, **4530** are logical servers on one or more physical servers. The hosted servers include an account (**4522**, **4532**) into which applications are installed, and account management system (EMS) (**4524**, **4534**) for managing the accounts, and databases (**4526**, **4536**) for supporting the hosted infrastructure.

[0394] In some embodiments, the databases **4536** can include relational database management systems such as PostgreSQL, MySQL, and optionally Oracle, Microsoft SQL, or any other Open Database Connectivity (ODBC) interfacing database may be used.

[0395] FIG. **46** illustrates a server system **4600** for a web application marketplace and hosting infrastructure. One or more servers **4602** hosts the marketplace application. The one or more servers include one or more processors **4604**. The processors include memory **4610** storing instructions for implementing the marketplace application. In some embodiments, the circled numbers of FIG. **46** correspond to steps shown in the flow diagram of FIG. **47**.

[0396] A marketplace application **4612** (executed in some embodiments by marketplace module **6122**, FIG. **61**) receives from a vendor a software application for distribution. The vendor can include a developer, coder, broker, or licensee with right to resell software. In some embodiments, the marketplace receives the application via an Internet (including in some embodiments a world wide web) connection **4618**. License terms **4622** are associated with the software application by a licensing module **6126** (FIG. **61**). The associating may include storing a license selection in a license manager, with a link/pointer to the account storing the respective software application. The software application is made available for distribution through the marketplace application, executed in some embodiments by the distribution module **6120** (FIG. **61**). Making the software application available for distribution can include displaying (via a listing manager) a summary of the software application, price, license terms, and a platform(s) or framework (s) on which the software application operates, in a catalog of software applications hosted by the marketplace application. The distribution may be initiated through the marketplace app, but the distribution and deployment process do not have to take place through the marketplace application.

[0397] Throughout this application, displaying means that the server sends data for display to a client associated with the user. Prior to display, the data for display may be formatted by the server prior to sending to the client associated with the user, or may be formatted by the client after receiving the data, or may include a combination of these operations.

[0398] In some embodiments, the marketplace application presents a description of the license terms to a user, via the user's web application or client **4616**. This presenting may be performed by the listing module **6122** (FIG. **61**). The description can be a full or brief description (e.g., open source, proprietary, source code, object file, repackaging, etc.) or an icon indicating a license type associated with the software application.

[0399] In response to a user selection of one or more software applications from the marketplace **4612**, the software application is deployed by deployer **4624** to one or more user accounts on one or more hosting servers **4630**. In some embodiments, a deployment confirmation is sent back to the marketplace **4612**. The hosting servers **4630** may be provided by the marketplace operator, or may be provided by the user. The hosting servers **4630** may be physical servers, or virtual servers. The deployment is performed in accordance with the license terms, executed by application deployment module **6150** (FIG. **61**). In some embodiments, the deploying is associated with a billing operation **4620**. In some embodiments, the one or more user accounts are accounts associated with the user who made the request to deploy the software application. Deploying includes at least one of the following: downloading the software application to the one or more user accounts, and/or enabling a flag associated with the software application in the one or more user accounts, enabling a license for the software application to the one or more user accounts, where the application itself stays stored at the marketplace server.

[0400] In some embodiments, a software vendor provides a license key to the marketplace application, whereby the marketplace application or a related application deploys/provisions the license key to a user account, and makes an API call to the software vendor at or after the time that the software application is licensed by the user. An application programming interface (API) is a set of declarations of the functions (or procedures) that an operating system, library or service provides to support requests made by computer programs. In some embodiments, the purpose of the API call is to inform the software vendor that a particular user or account has licensed the software application, which version is licensed, etc. In some embodiments, this information is beneficial to the software vendor when responding to technical support requests from the user.

[0401] In some embodiments, a software vendor may provide a license key associated with the software application to the marketplace application, and the license key is deployed to an account associated with a licensee. In some embodiments, the deployed license key enables the account to access or execute the software application. In some embodiments, the purpose of the license key is to enforce licensing terms.

[0402] In some embodiments, a software vendor may provide a license key update to the marketplace application, and

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 203

the license key update is deployed to an account associated with a licensee, where the license key update modifies an already installed license key. In some embodiments, this modification includes extending or shortening a period of time during which the account may access the software application. In some embodiments, this modification includes adding or removing features that the account may access in the software application. When the application is deployed to an account associated with a user, in some embodiments confirmation is provided to the user, and then the user can access and run the application deployed at hosting infrastructure **4630**.

[0403] In some embodiments, the hosting infrastructure servers are physically separate from the marketplace servers hosting the marketplace application. In some embodiments, the deploying of a software application is performed in response to a user request to deploy the software application. In some embodiments, the marketplace application determines a user account type, and based on the user account type, prepares to deploy (via the deployer) a software application to the user account.

[0404] In some embodiments, distribution to a user through the marketplace application **4612** is performed through a website (e.g., **4618**) associated with the marketplace application. In some embodiments, distribution to a user through the marketplace application includes distribution through a client associated with the marketplace application **4612**, where the client accesses the marketplace through a secure connection.

[0405] In some embodiments, the software application is packaged, executed in some embodiments by packager module **6136**, FIG. **61**, for distribution via the marketplace application **4612** hosted by the one or more servers **4602**. The packaging can include one or more of parsing code for the software application, performing quality control for the software application, testing the software, packaging the software application for distribution, and packaging an update to the software application or packaging an update to the licensing of an application. In some embodiments, the packaged software application is stored in an application repository **4614**.

[0406] In some embodiments, software applications are presented, described and made available at the marketplace application by a listing manager, executed by listing module **6124**, FIG. **61**. The listing manager performs one or more of storing listings (sale details for software applications), storing technical articles and information, and storing support resources (help, customer support, etc.). The listing manager may also provide one or more of a search function, a list of most popular software applications, a package deal for licensing a suite of software applications, a customer review, and other related functions. In some embodiments, the listing manager includes a store listing for licensing the software application. In some embodiments, the marketplace application stores, or stores links to, technical support for the software application. This technical support includes application notes, instructional articles, datasheets, frequently asked questions (FAQs), etc.

[0407] FIG. **47** illustrates a flowchart **4700** of operations for selecting and deploying a software application in a user account. A user navigates (**1**) the web (or accesses via a client), and visits the application marketplace (**4702**). The user selects an application from the marketplace (**2**), and checks out (**4704**). The application is provisioned (**3**) (or deployed) on demand to the hosting infrastructure (**4706**). In some embodiments, the hosting infrastructure installs the application and sends confirmation (**4**) back to the marketplace (**4708**). In some embodiments, a link is sent (**5**) from the marketplace to the user's application (**4710**). The user accesses the application (**6**) and the application validates the license with the licensing server (hosted at the hosting infrastructure) via their client or browser (**4712**).

[0408] FIG. **48** illustrates a system **4800** including a marketplace server **4802** and a hosting infrastructure **4810**. The hosting infrastructure includes one or more hosting servers **4820**, **4830**, **4840**. The one or more hosting servers include one or more deployed software applications **4822**, **4824**, **4826**.

[0409] A user accesses a marketplace **4806** via a web connection or client **4804**. The user selects and checks out a software application, as described. The software application is hosted at the one or more hosting servers, e.g., deployed software application **4822**, at server **4820**. The user accesses the deployed software through a client or web connection **4808**, as described. Each server may host one or more software applications associated with one or more user accounts. The servers may be physical servers or logical servers operating in one or more groups on a physical server. User accounts are managed by user accounts module **6156** (FIG. **61**).

[0410] In some embodiments, deploying an application (in some embodiments executed by or under the control of a deployer module **6152**, FIG. **61**) includes downloading the software application to the one or more user accounts. User account access is managed by user account access module **6172** (FIG. **61**). In some embodiments, deploying an application includes activating a flag associated with the software application in the one or more user accounts that indicates when the application is available to a user associated with the account. In some embodiments, the flag enables the software application for the user. In some embodiments, deploying an application includes activating a license for the software application in the one or more user accounts, which activation renders the software application available to a user associated with the corresponding account. In some embodiments, deploying an application includes providing the software application for hosting by a user on hosting servers associated with the user, the hosting executed by user hosting module **6174** (FIG. **61**).

[0411] FIG. **49** illustrates a server system **4900** for selecting license options in a software distribution marketplace. This figure shows operations at the server, under control of the licensing manager, where a vendor may associate one or more license terms with its software application. The server system includes one or more servers **4902**, hosting a marketplace application. The server system is configured to receive from a vendor **4910** a software application for distribution **4920**. The server system includes a licensing manager **4930** and is configured to perform under control of the license manager **4930** a number of operations **4920**-**4928** related to software vendor selection of license terms associated with the software application.

[0412] License manager **4930** is configured to generate license terms in response to a selection by the vendor from options provided by the marketplace application. The license terms are associated **4922** with the software application. In some embodiments, the license terms are stored in a licensing manager.

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 204

[0413] The server is configured to make the software application available for distribution (**4926**) through the marketplace application, in accordance with the license terms.

[0414] In some embodiments, the software application is deployed **4928** to one or more user accounts on one or more hosting servers, in accordance with the license terms. Deploying an application in accordance with the license terms means that conditions specified by the software vendor in the license terms (e.g., source code is to be hidden from the licensee) are enforced when deploying and when granting access to the software application. In some embodiments, a deployer enables or disables functions or options within the deployed software application, where this enabling or disabling is determined by the license terms provided by the software vendor.

[0415] In some embodiments, the deploying is in response to a payment associated with the one or more user accounts.

[0416] In some embodiments, the license terms are selected by a vendor **4910** of the software application. In some embodiments, the license terms are selected from a grid **4912** or set of options provided by the marketplace application. In some embodiments, the license terms include at least one of one or more open source licenses **4932**, closed licenses **4934**, object file licenses **4936**, source code licenses **4938**, and repacking licenses **4940**. An open source license **4932** may provide typical license terms for open source software, e.g., a GNU license or BSD license, where the application source code is available under terms that allow for modification and redistribution without having to pay the original author. A closed license **4934** (also known as a proprietary license) is one that does not allow access to source code. An object file license **4936** is one that allows a user to execute the software application, but not to access any source code or object files. The source code license **4938** is on that allows a user to access the source code of the licensed software application, including making changes to the source code, but only allows distribution of binary code. In some embodiments, a repacking license **4940** determines whether a user of the software application is permitted to repackage and redistribute the software application, in accordance with repacking terms **4952**. In some embodiments, the repacking license has an associated royalty. In some embodiments, the royalty is one selected from a wholesale royalty, a retail royalty, or a flat fee. In some embodiments, the vendor may upload its own term sheet(s) for its software applications, and this term sheet is provided with the software application.

[0417] In some embodiments, the license terms may address (limit or permit) redeployment of software to a user within the same organization as the licensing user. This might be the case where a system administrator licenses a software application, customizes it for her organization, and then redistributes the customized software to users within her organization. In some embodiments, the license terms may allow redeployment to the whole marketplace. This might be the case where a developer can license a first software application, write more code to add value to it, and resell the enhanced software application to the marketplace. In this situation, a licensee could be required to remit a royalty to the vendor of the first software application or to the marketplace administrator, or to some combination thereof.

[0418] In some embodiments, the license manager **4930** determines user permissions for installation, activation, and access to features of applications. In some embodiments, the license options provided by the marketplace application include an option to use license terms **4914** supplied by the vendor. These vendor-supplied terms **4914** may be considered as 'hard' terms (i.e., a fixed termsheet provided by the software vendor, rather than terms generated by the marketplace application in response to selections of terms by the user), since they are not generated by the license manager in response to a selection from the standard license options.

[0419] In some embodiments, the license manager **4930** includes a transaction reporter **4950** (executed in some embodiments by transactions report module **6132**, FIG. **61**) to display licensing events for a respective software application made available for distribution through the marketplace application. In some embodiments, this transaction reporter shows transactions going through the marketplace for a respective software application, including installation status, user check-in status, payment status etc. In some embodiments, the transaction reporter displays the licensing events to a respective vendor associated with the respective software application.

[0420] In some embodiments, a price associated with the software application is stored at a licensing manager separately from the software application. In some embodiments, the price is dynamically adjusted by the licensing manager in response to a selection by the software vendor, or by the marketplace administrator, or by usage metrics triggers embedded in the software application itself. In some embodiments, an adjustment to the price is provided to the marketplace, which updates pricing information to charge the adjusted price for licensed software applications.

[0421] In some embodiments, at least one selected from the group consisting of access duration, features, and price, is dynamically adjusted by the licensing manager in response to a selection by the software vendor, or by the marketplace administrator. The access duration includes a trial period for the software application, or other promotional offers. The features includes access to features (including new or premium features) within the software application. The price is the price associated with the licensed software application of features thereof. Vendors can also choose, bandwidth allowed, disk space, users, record table max limits and/or if ads are displayed in the application, among other things.

[0422] FIG. **50** illustrates a server system **5000** for performing dynamic billing in a software distribution marketplace. This figure shows operations at the server, under control of the billing manager, where a vendor may associate one or more prices with its software application. The server system includes a billing manager **5030**, and is configured to perform under control of the billing manager **5030** a number of operations (including **5010**-**5016**, indicated on FIG. **50** as circled numbers **1** to **4**) related to user selection of license terms associated with the software application.

[0423] The server system includes one or more servers **5002**, hosting a marketplace application **4806** (FIG. **48**) and a billing manager **5030**. The server system is configured to receive from a vendor a software application for distribution **5010**, and store it in an application repository, executed by application repository module **6134** (FIG. **61**). The system is configured to associate a licensing price (stored in the billing manager **5030**) with the software application **5012**, where the license price is controlled by billing manager **5030**, executed by billing module **6130** (FIG. **61**). The licensing management is executed by licensing module **6126** (FIG. **61**). The vendor of the software application provides a price per license **5020**, which may be stored by the server at the billing manager

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 205

5030. This price per license may be changed dynamically (i.e. changed 'on the fly') by the vendor 5020 or by an administrator 5022 of the software distribution marketplace. The billing manager 5030 updates the prices associated with each license type (if more than one) for a software application.

[0424] In some embodiments, a first price 5042 is associated with an open source license 5032, a second price 5044 is associated with a closed license 5034, a third price 5046 is associated with an object file 5036, a fourth price 5048 is associated with a source code license 5038, and a fifth price 5050 is associated with a repacking license 5040. In some embodiments, there are more or fewer prices associated with one or more of the license types. In some embodiments, a subscription or 'flat fee' price 5052 is supported by the billing manager. In some embodiments, the server 5002 receives a payment 5014 for licensing the software application. In some embodiments, the billing manager supports payment processing (for paid licenses). In some embodiments, the billing manager supports registration (where no payment required for a license) 5054.

[0425] During the licensing process, (e.g., before a user pays to license a software application), license terms associated with the software application, or a link to those license terms, are displayed 5018 to the user. In some embodiments, the license terms are provided to the user before executing the software application. In some embodiments, the license terms are provided as click-wrap license terms. The server is configured to deploy 5016 the licensed software application to an account associated with the user.

[0426] FIG. 51 illustrates a server system 5100 for managing an application repository and deployer in a software distribution marketplace. This figure shows operations at the server, under control of the application repository and/or deployer, where software applications provided by a software vendor may be deployed to a user account associated with a user. Server 5102 is configured to receive a software application from a software vendor for distribution 5110. An application packager 5120 (executed in some embodiments by a packager module 6136, FIG. 61) packages 5112 the software application for distribution. An application deployer 5150 deploys the software application to a user account 5116. The packaged application is stored 5114 in an application repository. In some embodiments, the application deployer serves as a gateway between the user account and the application repository 5140.

[0427] In some embodiments, the server 5102 hosts the deployed software application for one or more user accounts. The hosting includes executing the deployed software on behalf of the associated users. In some embodiments, the software application is hosted at a server separate from the server 5102.

[0428] In some embodiments, the application packager 5120 prepares updates 5122 to previously deployed software applications 5214. In some embodiments, this preparation included one or more of testing the software application for functionality and/or performance to ensure quality, providing technical support material (e.g., a manual, product application notes, etc.), and specifying a language and framework associated with the software application.

[0429] In some embodiments, the previously deployed function 5124 is required for the update 5122 to operate. Such an update is sometimes referred to as a 'mini application' and may include plugins or add-on features that can be bought from the store. In some embodiments, the 'mini application' may include updates to the applications.

[0430] In some embodiments, the application packager 5120 prepares updates to standalone distributions 5126. A standalone distribution is one that does not require a previously deployed function to operate. In some embodiments, the standalone distribution includes a software application and one or more patches to the software application 5128.

[0431] In some embodiments, the application packager 5120 prepares applications or updates for distribution 5130 by performing a quality assurance test 5132. This test may be automatic or may involve some human input. The application packager 5120 includes technical support material 5134 with the packaged application (e.g., a readme file). The application packager 5120 may include instructions to specify what language, framework, database, supporting packages and libraries, or operating system the software application is compatible with 5136.

[0432] In some embodiments, the application repository 5140 stores a plurality of packaged applications 5142, 5144, to 5146. These packaged applications are those applications that are ready for distribution and deployment to a user account.

[0433] In some embodiments, the application deployer 5150 retrieves a packaged application from the repository 5140 if it does not exist in a local cache and then deploys it to a user account (5152, 5116). The deployer can deploy an application by a push method 5154 (where a user does not have any choice about whether or not to receive the deployed update or application), by a pull method 5156 (where the user chooses to retrieve the deployed update), or by a hybrid method 5158 (which may be a combination of the push and pull methods, or another method). The application deployer performs deployment in accordance with license terms 5160.

[0434] In some embodiments, the application deployer 5150 generates a new user account compatible with the software application and deploys the software application to the new user account. The account type can include one or more of operating system, language, or framework associated with the user account.

[0435] FIG. 52 illustrates a server system 5200 for performing syndicated deployment of a software application across a network, executed by syndicated deployment module 6160 (FIG. 61). Server 5202 is configured to receive a request from syndicated server 5230, across a network 120, as described. In some embodiments, the request is received at marketplace 5210, including at deployer 5214. The server identifies one or more user accounts associated with the request. The server (including deployer 5214) checks licensing terms and permissions 5212, to determine if the syndicated server, including syndicated server software distribution 5232 and syndicated server user accounts 5234, have permission to receive the deployed software application, and if the syndicated deployment is in accordance with the license terms.

[0436] In some embodiments, deploying an application includes providing through an application deployer, across a network to one or more user accounts, a software application stored at the application repository. The deployer accesses repository 5216 and retrieves an application ready to deploy 5218. This application is deployed to syndicated server user account 5234, in accordance with the license terms.

[0437] In some embodiments, deploying an application includes presenting the software application for deployment to a user or to a syndicated server associated with the user. In

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 206

these embodiments, the syndicated server deploys the presented software application into a user account associated with the syndicated server. A syndicated server is a server that is logically separate from the marketplace server. In some embodiments, the syndicated server is also physically separate from the marketplace server and is controlled by someone other than the controller of the marketplace and hosting servers. In some embodiments, the syndicated server primarily provides services other than the licensing of software applications in a marketplace. In some embodiments, the syndicated server offers to its users the ability to license software applications from the marketplace by providing its users with an interface (in some embodiments, implemented using an API) by which to browse and/or license software applications from the marketplace. In some embodiments, a licensed application is deployed to an account associated with the user, hosted by or under the control of the syndicated server.

[0438] In some embodiments, the software application is made available for distribution through the syndicated server. In this embodiment, the syndicated server acts as a marketplace front for the software application, and the application is licensed by a user through the syndicated server.

[0439] In some embodiments, deploying an application includes selecting one of a plurality of software applications, compatible with the one or more user accounts, from the application repository. The selected software application is deployed to an account associated with the user, hosted by the hosting infrastructure server.

[0440] In some embodiments, the application repository stores software applications in any one or more of a number of states, including at least one state of a ready to deploy state, an undergoing quality assurance state (undergoing testing **5220**), a ready to submit for quality assurance state, and an unfinished state. The ready to deploy state means that the software application is fully tested and is ready for licensing to a user. Only software applications in the ready to deploy state can be listed on the marketplace and licensed to a user, and thus only software applications in this state can generate revenue. The undergoing quality assurance state means that the software application is being tested and verified for quality, and is not yet ready for licensing to a user. The ready to submit for quality assurance state means that the software application code is written, but testing and quality assurance has not yet been performed on it. The unfinished state means that the software application is still in development.

[0441] FIG. **53** illustrates an embodiment of a server system **5300** for packaging software applications for distribution to a user account. The system **5300** includes a packager **5320**, executed in some embodiments by a packager module **6136** (FIG. **61**).

[0442] The packager receives as input software application files **5310**, and prepares them for deployment. In some embodiments, the packager also receives data files, local install scripts **5312** and auxiliary software packages **5314** associated with the software application files **5310**. The packager prepares one or more of the files **5310**, **5312** and **5314**, and generates a packaged application **5330**, which is stored in application repository **5342** associated with marketplace application **5340**. Upon licensing by one or more users **1** to N, the packaged application **5344** is deployed to one or more accounts **1** to N associated with the users **1** to N, on hosting application servers **5350-1** to **5350**-N. This deploying is performed by the application deployer.

[0443] FIG. **54** illustrates an alternate embodiment of a server system **5400** for packaging updates to software applications for distribution to a user account. The system **5400** includes a packager update tool **5420**, which in some embodiments is executed by the packager module **6136** (FIG. **61**).

[0444] The packager receives as input updated software application files **5410**, updated data files, local install scripts **5412**, and updated auxiliary software packages **5414** prepares them for deployment, and generates an updated (patched) application **5430**. In some embodiments, the updated application (patch) **5446** is stored in the application repository **5442** associated with marketplace application **5440**, for deployment to one or more servers **5450-1** to **5450**-N. The updated application (patch) is deployed to servers that already have the original packaged application (e.g., version 1) **5444** installed.

[0445] FIG. **55** illustrates an alternate embodiment of a server system **5500** for packaging updates together with software applications for distribution to a user account. System **5500** includes packager **5520**, executed by packager module **6136** (FIG. **61**). Packager **5520** packages both software applications and updates files for the software applications separately or together for distribution.

[0446] The packager receives as input software application files **5510**, updated data files **5512**, and updated auxiliary software packages **5514**; prepares them for deployment either together or separately; and generates a packaged new version of an application **5530**. In some embodiments, a packaged new version of an application is stored in the application repository **5542** associated with the marketplace application **5440**, for deployment to one or more servers **5550-1** to **5550**-N. The packaged new version of the application **5549** is deployed by the application deployer to one or more user accounts associated with hosting infrastructure servers. In some embodiments, the application repository **5542** stores the original packaged application **5544**, and updates **5546** to **5548** to the packaged application. In some embodiments, when the packaged new version of the application **5549** is placed in the application repository **5542**, the older version **5544**, and updates **5546** to **5548**, may be removed from the application repository.

[0447] FIG. **56** illustrates a method **5600** of licensing a software application from a marketplace, for deployment to a user account associated with a user, hosted at a hosting infrastructure server. In this scenario, the licensing process includes receiving a selection from the user for a software application from the marketplace and adding the application to a cart associated with the user, receiving the user's request to check out the cart and thereby the user's request to license the software application, providing the user with terms of use associated with the software application, and receiving the user's agreement to the terms of use (in some embodiments, receiving a payment from the user). In some embodiments, upon licensing of the software application by the user, the software application is deployed to an account associated with the user.

[0448] At one or more servers hosting a marketplace application **5620**, one or more software applications **5622** to **5624** are made available for distribution through the marketplace application. The marketplace server receives a user request to license the software application. The marketplace server provides the software application for deployment **5648** to one or

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 207

more user accounts, hosted on the one or more servers, or hosted at a hosting infrastructure server, as described regarding FIG. **44**.

[0449] In some embodiments, providing the software application for deployment includes providing the software application across a network for deployment at one or more servers associated with the user, hosting the one or more user accounts.

[0450] In some embodiments, the user request includes a selection by the user to add the software application to a cart **5640** associated with the user, and to check out **5642** the cart.

[0451] In some embodiments, prior to deploying or prior to receiving the user request, the marketplace server presents a description of license terms of use associated with the software application to the user **5644**.

[0452] In some embodiments, the license terms are specified by a vendor **5614** associated with the software application. The license terms may be specified before the software application is presented for deployment, or the license terms may be specified following deployment. In some embodiments, the license terms may evolve after deployment. In some embodiments, specifying the license terms includes selecting (by the vendor) the license terms **5614** from a plurality of options provided by a licensing engine **5630** associated with the one or more servers hosting a marketplace application. In some embodiments, the licenses are stored **5632**, **5634** at the licensing engine **5630**. In some embodiments, the licensing engine **5630** validates that the user request to license the software application complies with the license terms.

[0453] In some embodiments, the request to license the software application includes a payment **5646** (e.g., credit card or debit card payment, or bank transfer). In some embodiments, the request may be associated with a prospective future payment (trial period). The prospective future payment may include receiving a credit card number, but not billing the credit card until some time in the future.e. g., at the expiration of the trial period.

[0454] In some embodiments, prior to presenting an application for deployment, the vendor builds **5610** and packages **5612** the software application **5616**. In some embodiments, the software application is a web-based software application, executed at one or more servers.

[0455] FIG. **57** illustrates a system **5700** for providing security for software applications, deployed to a user account. One or more servers **5702** hosts a marketplace application **5710**. The marketplace application includes a billing manager **5712**. The marketplace application is accessed by a web connection or client **5714**. The deployed application is accessed by a user's web application **5720**, having an associated user identifier **5722**.

[0456] The server **5702** includes a security application **5730**, executed by deployment security module **6158** (FIG. **61**). In some embodiments, the security application **5730** verifies that only a licensed set of features and/or number of copies of a deployed software application is running at a time **5732**. These requirements may be provided by the software vendor via the licensing manager (described in FIG. **49**).

[0457] In some embodiments, the security application **5730** includes a payment verifier **5736** to compare a user identifier **5722** associated with the one or more user accounts and an application id (e.g. **5750**, **5760**) associated with the deployed application against the billing manager **5712** to verify that a valid payment has been recorded. In some embodiments, this verification is performed prior to executing the deployed application. The security application verifies that both the user and application are associated with a valid transaction before allowing a user to execute the software application.

[0458] In some embodiments, the security application **5730** includes a permission verifier **5738** to verify that the one or more user accounts have permission to execute the deployed application, and in the event of a verification failure, to warn the user. The user account permissions are executed by permissions module **6128** (FIG. **61**). In some embodiments, the security application **5730** includes periodic verifier **5740** to periodically (e.g., daily, weekly, monthly etc.) repeat one or more of the verification steps described. Following one or more verification failures, a shutdown application **5742** warns the user **5744**, and/or prevents **5746** (in the event of multiple failures) the one or more user accounts from executing the deployed application. The plurality of failures includes factors such as the number of failures, the period of time over which failures occur, IP addresses from which attempted logons to the application are made, etc. In some embodiments, the shutdown application **5742** prevents access by the user to data stored at the one or more servers, upon determining that the one or more user accounts has been disabled. The user access is executed by access module **6170** (FIG. **61**). The account enablement verifier **5734** verifies that a user account has permission to access the deployed software applications (e.g., **5752**, **5762**). In some embodiments, the user permission/access is made available to the application vendor through the licensing API to enable custom messaging and actions by the vendor. Making available means that a list of privileges or permissions associated with a the user is provided to the software application vendor so the vendor can understand what terms the user has agreed to when licensing the software. This can assist the software vendor in resolving a user's problems. In some embodiments, when a user is denied access, the user is redirected to the vendors marketplace storefront listing to resolve the issue.

[0459] In some embodiments, the security application **5730** includes instance verifier **5732** to check for multiple instances of the software application being simultaneously executed by the one or more user accounts. In some embodiments, for some software applications (e.g., AOP) multiple logons to a user account are permitted, but only one logon may use (execute) a software application associated with the user account at a time. As an example, a user can stay logged into her account at work, but go home and access the account from her home PC. Since she is using her work PC to execute applications while she is at home, she may execute a software application while logged into her account from her home PC.

[0460] If the security application **5730** determines that the user account is authorized to access a deployed software application, a confirmation **5772** is provided to a hosted application execution engine **5770** for running deployed software. This engine executes one or more of the hosted deployed software applications **5752**, **5762**, hosted on one or more servers **5754**, **5764**.

[0461] FIG. **58** illustrates a system **5800** for allowing multiple logons by a user **5860** to a software application, deployed to a user account. In an example, a user may require multiple logons if he logged onto the software application from his work computer, and left work and traveled home forgetting to log out. If the user wants to access the software application from his home computer (while still logged on from work, having forgotten to log out) then the user may log

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 208

in from home (thus creating a simultaneous logon situation). In another example, a user may be logged on to the software application from his desk computer, and while in a meeting at another location, may want to access the software application using a handheld device (thus creating a simultaneous logon situation). One or more servers **5805** hosts a licensing engine **5180**. The licensing engine includes a simultaneous logon manager **5811**, executed by user account access module **6172** (FIG. **61**). In some embodiments, the one or more servers **5805** also host a first **5814** and a second **5186** deployed software applications, where a user has licenses to the first and second deployed applications.

[0462] A first client system **5820** (e.g., an office PC) executes a browser **5822** in which a first instance **5824** of first software application **5814** is open and logged in. The first client system also includes an account management system (EMS) **5828**, a software stack **5830**, and an operating system (OS) stack **5832**. A second client system **5840** (e.g., a home PC) executes a browser **5842**, in which a second instance **5844** of first software application **5814** is open and logged in. The second client system also includes an account management system (EMS) **5848**, a software stack **5850**, and an OS stack **5852**.

[0463] When a client **5820** attempts to access a deployed application **5814** (associated with a user account), the licensing engine **5810** and logon manager **5811** determine if the client **5820** has permission to access the application, and also determine if another client is accessing that application, associated with the user account. In some embodiments, the logon manager **5811** will only permit (depending on the license terms specified by the software vendor via the licensing manager, described in FIG. **49**) a single logon to an application associated with a user account at a time. In some embodiments, the logon manager **5811** will permit a plurality of logons (e.g., by clients **5820** and **5840**) to an application associated with a user account, if the licensing terms so permit. In some embodiments, the login manager **5811** will permit multiple logons, as long as only one client system is operating the application at a time. Operating means providing input to the application and/or receiving information from the application.

[0464] FIG. **59** illustrates an exemplary user access control interface **5900** for controlling user access to a software application, deployed to a user account. This figure shows operations at the marketplace server, under control of the licensing manager, where a system administrator (or in some embodiments, a vendor) may view and/or change license terms associated with the software application and associated with users of those applications. This is useful because it gives the administrator a 'birds eye' view of licensing permissions and activity within the system.

[0465] A plurality of applications **5902-1** to **5932**-N are shown on the left hand column. Associated with each application is one or more users **5920-1** to **5920**-N, and **5934-1**. For each user, a status indicator **5904** determines if the user is active (i.e., using) that particular application. An administrator status **5906** determines if the user has administrator privileges. An AOP status **5908** indicates if a user has permission to run an application offline. A source code access status **5910** determines whether a user is allowed to view or edit source code associated with an application. By changing the status buttons or indicators, an administrator or vendor can change the permissions associated with an application or with a user. Other control **5912** determines if a user can access other features, in accordance with a particular embodiment.

[0466] FIG. **60** illustrates an exemplary user access control interface **6000** for controlling user access to a software application, deployed to a user account. One or more software applications **6002** through **6004** are shown. For each software application, one or more users **6010-1**, **6010-2** through **6010**-N may access the software application, where this access is controlled by this interface.

[0467] FIG. **61** is a block diagram illustrating a server system **6100** configured to host a software marketplace and licensing system. The system **6100** generally includes one or more processing units (CPU's) **6102**, one or more network or other communications interfaces **6104**, memory **6110**, and one or more communication buses **6108** for interconnecting these components. The communication buses **6108** may include circuitry (sometimes called a chipset) that interconnects and controls communications between system components. The system **6100** may optionally include a user interface, for instance a display **6106** and input device (e.g., a keyboard and/or mouse) **6105**. Memory **6110** may include high speed random access memory such as DRAM, SRAM, DDR RAM or other random access solid state memory devices; and may include non-volatile memory, such as one or more magnetic disk storage devices, optical disk storage devices, flash memory devices, or other non-volatile solid state storage devices. Memory **6110** may include mass storage that is remotely located from the central processing unit (s) **6102**.

[0468] Memory **6110**, or alternately the non-volatile memory device(s) within memory **6110**, comprises a computer readable storage medium. In some embodiments, memory **6110** stores the following programs, modules and data structures, or a subset thereof:

- **[0469]** an operating system **6111** that includes procedures for handling various basic system services and for performing hardware dependent tasks;
- **[0470]** a network communication module **6112** that is used for connecting the server **6100** to other computers via the one or more communication network interfaces **6104** (wired or wireless) and one or more communication networks, such as the Internet, other wide area networks, local area networks, metropolitan area networks, and so on;
- **[0471]** a distribution module **6120** that is used to market and distribute software applications to users of the software marketplace and licensing system;
- **[0472]** a marketplace module **6122** that provides an electronic marketplace where software vendors can display, and users can select, software applications for licensing;
- **[0473]** a listing module **6124** that is used to generate and display software application products within an electronic marketplace;
- **[0474]** a licensing module **6126** that controls the distribution of software applications, with license terms selected or provided by a software vendor;
- **[0475]** In some embodiments, a permissions module **6128** that enforces license terms during deployment;
- **[0476]** a billing module **6130** that receives and processes payment from a user associated with a licensed software application;
- **[0477]** a transaction report module **6132** for reporting on distribution and licensing activity taking place within the marketplace;

[0478] an application repository module **6134** for storing and managing software applications ready for distribution;

[0479] a packaging module **6136** for preparing software applications and updates for distribution;

[0480] a deployment module **6150**, for managing deployment of software applications to a user account, and managing hosting of the software application;

[0481] a deployer module **6152** (in some embodiments part of the deployment module **6150**) for deploying a software application to a hosted user account, or to a user-controlled server for hosting the account;

[0482] a hosting module **6154** for hosting the software application in a hosted user account and for executing the software application;

[0483] a user account module **6156** for managing user accounts, permissions and access;

[0484] a security module **6158** for managing deployment and ensuring that a software application is deployed in accordance with license terms, to authorized users, at authorized servers;

[0485] a syndicated deployment module **6160** for performing syndicated deployment (in some embodiments, across a network) in response to a request from a syndicated server;

[0486] an access module **6170** for controlling access to deployed applications and to the marketplace;

[0487] a user account access module **6172** (in some embodiments, part of user account module **6156** or access module **6170**) for controlling user access to hosted applications;

[0488] a user hosting module **6174** (in some embodiments, part of a hosting module **6154**) for hosting the deployed software application at a server controlled by the user;

[0489] a marketplace database management module **6180** for managing a marketplace database;

[0490] a licensing database management module **6182** for managing a licensing database;

[0491] a deployment database management module **6184** for managing a deployment database;

[0492] a billing database management module **6186** for managing a billing database; and

[0493] a user account database management module **6188** for managing a user account database.

[0494] In some embodiments, the database management modules **6180**, **6182**, **6184**, **6186**, **6188** can be separate or combined into one or more groups.

[0495] Each of the above identified elements may be stored in one or more of the previously mentioned memory devices, and corresponds to a set of instructions for performing a function described above. The above identified modules or programs (i.e., sets of instructions) need not be implemented as separate software programs, procedures or modules, and thus various subsets of these modules may be combined or otherwise re-arranged in various embodiments. In some embodiments, memory **6110** may store a subset of the modules and data structures identified above. Furthermore, memory **6110** stores additional modules and data structures not described above.

[0496] Although FIG. **61** shows a server system for software application distribution, FIG. **61** is intended more as functional description of the various features that may be present in a set of servers than as a structural schematic of the embodiments described herein. In practice, and as recognized by those of ordinary skill in the art, items shown separately could be combined and some items could be separated. For example, some items shown separately in FIG. **61** could be implemented on single servers or single items could be implemented by one or more servers. The actual number of servers used to implement a software application distribution system and the way in which features are allocated among them will vary from one implementation to another, and may depend in part on the amount of data traffic that the system must handle during peak usage periods as well as during average usage periods.

[0497] FIG. **62** illustrates a block diagram of a client system **6200** for accessing a software marketplace and licensing system, hosted at one or more servers. The client system **6200** generally includes one or more processing units (CPU's) **6202**, one or more network or other communications interfaces **6204**, memory **6210**, and one or more communication buses **6208** for interconnecting these components. The communication buses **6208** may include circuitry (sometimes called a chipset) that interconnects and controls communications between system components. The system **6200** also includes a user interface, for instance a display **6206** and input device (e.g., a keyboard and mouse) **6205**. Memory **6210** may include high speed random access memory such as DRAM, SRAM, DDR RAM or other random access solid state memory devices; and may include non-volatile memory, such as one or more magnetic disk storage devices, optical disk storage devices, flash memory devices, or other non-volatile solid state storage devices. Memory **6210** may include mass storage that is remotely located from the central processing unit(s) **6202**.

[0498] Memory **6210**, or alternately the non-volatile memory device(s) within memory **6210**, comprises a computer readable storage medium. In some embodiments, memory **6210** stores the following programs, modules and data structures, or a subset thereof:

[0499] an operating system **6211** that includes procedures for handling various basic system services and for performing hardware dependent tasks;

[0500] a network communication module **6212** that is used for connecting the client **6200** to a server hosting software applications, via the one or more communication network interfaces **6104** (wired or wireless) and one or more communication networks, such as the Internet, other wide area networks, local area networks, metropolitan area networks, and so on;

[0501] a receive and process user input module **6214** for receiving input from input device(s) **6205**, such as a keyboard and mouse, and interpreting that input to control a client/browser;

[0502] a client/browser module **6215** (including browser engine module **6216**) for providing a window by which a user accesses the hosted software application, and through which the user (or in some embodiments an administrator) manages the hosted software application;

[0503] accessible through the client/browser module **6215**: a distribution module **6220**, application deployment module **6250** and access module **6270**, each with constituent modules, and database management modules **6280** through **6288**, corresponding to modules in FIG. **61** as described earlier;

[0504] accessible through the client/browser module **6215**: a cart checkout/module **6290** for allowing a user to

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 210

place a software application in the cart, and checking out the cart to license the software application;

[0505] accessible through the client/browser module **6215**: a process payment module **6292** for processing a payment associated with the licensed software application;

[0506] accessible through the client/browser module **6215**: a select license module **6294** for selecting a license associated with a software application, prior to licensing the application;

[0507] accessible through the client/browser module **6215**: a submit license terms module **6296** for a vendor to select license terms associated with the software application, or to submit custom license terms provided by the vendor (e.g., a text file); and

[0508] auxiliary services module(s) **6298** for providing other features or services associated with the client.

[0509] Flowcharts for Marketplace and Licensing

[0510] FIG. **80** is a flowchart representing a server method **8000** for distributing a software application, according to certain embodiments of the invention. Server method **8000** may be governed by instructions that are stored in a computer readable storage medium and that are executed by one or more processors of one or more servers. Each of the operations shown in FIG. **80** may correspond to instructions stored in a computer memory or computer readable storage medium. The computer readable storage medium may include a magnetic or optical disk storage device, solid state storage devices such as Flash memory, or other non-volatile memory device or devices. The computer readable instructions stored on the computer readable storage medium are in source code, assembly language code, object code, or other instruction format that is interpreted by one or more processors.

[0511] At one or more servers hosting a marketplace application (e.g., FIG. **129**, marketplace storefront; FIG. **155**, user marketplace), a software application is received from a vendor for distribution (**8002**). License terms (e.g., FIG. **133**, license terms) are associated with the software application (**8004**). In some embodiments, the license terms are received from the vendor of the software application, and are stored in a database (e.g., by licensing database management module **6182**, FIG. **61**) with a pointer to the vendor's software application. Associated means that when the software application is selected for licensing by a user, the vendor license terms corresponding to that vendor's software application are used during licensing and distribution of the software application. In some embodiments, a vendor may have different sets of license terms per software application.

[0512] The software application is made available for distribution (e.g., FIGS. **129**, **155**) through the marketplace application (**8006**). The software application is deployed (e.g., FIG. **175**, "My Installations") to one or more user accounts on one or more hosting servers, in accordance with the license terms.

[0513] In some embodiments, the one or more hosting servers (e.g., FIG. **148**, developer hosting accounts; FIG. **149** hosting sub-domains) are physically separate from the one or more servers hosting the marketplace application (**8026**).

[0514] In some embodiments, the deploying is performed after a user request (e.g., FIGS. **166**, **167**, **168**, download and install) to deploy the software application (**8101**). In some embodiments, the deploying includes downloading the software to one or more user accounts (**8012**). In some embodiments, deploying includes activating a flag associated with the software application in the one or more user accounts (**8014**). In some embodiments, the flag enables the software application for the user account (**8016**). In some embodiments, the deploying includes activating a license for the software application in the one or more user accounts (**8018**).

[0515] In some embodiments, the deploying includes providing the software application for hosting by a user on hosting servers (e.g., FIG. **148** developer hosting account, FIG. **149** sub-domains associated with an account) associated with the user (**8020**).

[0516] In some embodiments, distribution to a user through the marketplace application includes through a website (e.g., FIG. **155**, marketplace storefront) associated with the marketplace application (**8022**). In some embodiments, distribution to a user through the marketplace application includes distribution through a client associated with the marketplace application (**8024**).

[0517] In some embodiments, the deploying is performed (e.g., FIG. **175**, "My Installations") by an application deployer (**8028**), such as the application deployer described in FIG. **51**, and/or the application deployer module **6152** (FIG. **61**).

[0518] In some embodiments, the deployed software application is hosted for the one or more user accounts (**8030**).

[0519] FIG. **81** is a flowchart **8100** continuing the server method **8000** of FIG. **180**.

[0520] In some embodiments, a description of the license terms (e.g., FIG. **158**, accept terms) is presented to a user (FIG. **81**, **8114**).

[0521] In some embodiments, the software application is packaged (e.g., FIG. **154**, **173**, My Packaged Applications) for distribution via the marketplace application hosted by the one or more servers (**8102**). In some embodiments, a packaged software application is stored in an application repository (**8104**). In some embodiments, the packaging includes preparing an update (e.g., FIG. **103**, update **10304**) to a previously deployed software application, where the update requires the previously deployed software application to function (**8106**). This scenario might be where a patch update (set of fixes to a previously deployed application) is deployed. A patch generally requires the earlier deployed application to function, since the patch only changes a small portion of the deployed application, and thus the patch is not a standalone application.

[0522] In some embodiments, the packaging includes preparing a standalone distribution (e.g., FIG. **103**, update **10302**) for a software application (**8110**). In some embodiments, the standalone distribution includes a software application and one or more patches to the application (**8112**). In some embodiments, the update is deployed to the one or more user accounts selected from the group consisting of (e.g., FIG. **142**, Product Upgrade) a push method, a subscription (pull) method, and a hybrid method, in accordance with the license terms (**8108**).

[0523] In some embodiments, the making available (e.g., FIG. **130**, Product/Service Basic Information) is performed by a listing manager (**8116**), executed by listing module **6124** (FIG. **61**). In some embodiments, the listing manager includes a store listing (e.g., FIGS. **130-153**, store listing details/setup, FIG. **155** storefront) for licensing the software application (**8118**). In some embodiments, the marketplace application includes technical support (e.g., FIGS. **174**, **177-181**, support and knowledge base) for the software application (**8122**). By making available, the listing manager per-

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 211

forms one or more of storing and presenting software application listings (sale details for software applications), storing technical articles and information, and storing support resources (help, customer support, etc.). The listing manager may also provide one or more of a search function, a list of most popular software applications, a package deal for licensing a suite of software applications, a customer review, and other related functions.

[0524] In some embodiments, making the software application available for distribution can be done by determining a user account type, and based on the user account type, preparing to deploy a software application to the user account, or generating a new user account (e.g., FIG. **169**, developer accounts) compatible with the software application and preparing to deploy the software application to the new user account (**8120**).

[0525] FIG. **82** is a flowchart representing a server method **8200** for licensing a software application, according to certain embodiments of the invention. The server method **8200** may be executed by licensing module **6126** (FIG. **61**) and as described earlier with regard to FIG. **49**.

[0526] At one or more servers, hosting a marketplace application, a software application is received from a vendor for distribution (**8202**). License terms (e.g., FIG. **132**, license selection) are generated (**8204**) in response to a selection by the vendor from options provided by the marketplace application. The license terms are associated with the software application (**8206**). The software application is made available for distribution through the marketplace application (e.g. FIG. **155**, storefront), in accordance with the license terms (**8208**).

[0527] In some embodiments, the software application is deployed (e.g., FIG. **175**, My Installations) to one or more user accounts on one or more hosting servers, in accordance with the license terms (**8210**). For example, the license terms can influence an application in the following way. Deploying an application in accordance with the license terms means that conditions specified by the software vendor in the license terms (e.g., source code is to be hidden from the licensee) are enforced when deploying and when granting access to the software application. In some embodiments, a deployer enables or disables functions or options within the deployed software application, where this enabling or disabling is determined by the license terms provided by the software vendor.

[0528] In some embodiments, the deploying is in response to a payment (e.g., FIG. **159**, billing history) associated with the one or more user accounts (**8212**).

[0529] In some embodiments, license terms include at least one of (e.g., FIG. **132**, part four) an open source license, a closed license, a source code license, an executable (object file) license, and a repacking license (**8214**). In some embodiments, the repacking license (e.g., FIG. **132**, part three) determines whether a user of the software application is permitted to repackage and redistribute the software application (**8216**). In some embodiments, the repacking license has an associated royalty (**8218**). In some embodiments, the associated royalty is one selected from the group consisting of a wholesale royalty, a retail royalty, and a flat fee (**8220**). In some embodiments, the license manager determines user permissions (e.g., FIG. **132**, part four) for installation, activation, and access to features of applications (**8222**). In some embodiments, the options provided by the marketplace application includes an option to use license terms supplied by the vendor (**8224**).

[0530] In some embodiments, licensing events (e.g., FIG. **153**, transaction report) for a respective software application are made available for distribution through the marketplace application (**8226**). In some embodiments, the license events are displayed (e.g., FIG. **151**, store report) to a respective vendor associated with the respective software application (**8228**). In some embodiments, the license terms are stored in a licensing manager (**8230**).

[0531] FIG. **83** is a flowchart **8300** continuing the server method **8200** of FIG. **82**.

[0532] In some embodiments, a price (e.g., FIG. **132**, part two; FIG. **144** pricing grid) associated with the software application is stored at a licensing manager separately from the software application (**8302**). In some embodiments, the price is dynamically adjusted by the licensing manager in response to a selection by the software vendor (**8304**). In some embodiments the vendor selects a price by entering a price (e.g., price in US dollars, Euros, Yen, or any other currency) in the licensing manager **4930** (FIG. **49**) or billing manger **5030** (FIG. **50**). In some embodiments, the vendor selects a price by choosing a price from a set of options provided by the marketplace. In some embodiments, at least one selected from the group consisting of access duration, features, and price, is dynamically adjusted by the licensing manager in response to a selection by the software vendor (**8306**).

[0533] In some embodiments, the one or more user accounts are stored separately from the marketplace application (**8308**).

[0534] In some embodiments, a payment (e.g., FIG. **159**, billing history) associated with the software application is processed (**8310**). In some embodiments, prior to processing a payment, a promotional code is received from the user, and the payment is processed based on the promotional code (**8312**). In some embodiments, a record of the processed payment is stored in a billing record (**8314**).

[0535] In some embodiments, prior to executing the deployed application, a user identifier associated with the one or more user accounts and an application id associated with the deployed application are compared against a billing manager to verify that a valid payment has been recorded (**8316**). In some embodiments, the system verifies that the one or more user accounts has permission to execute the deployed application, and in the event of a verification failure, warns the user (**8318**). In some embodiments, the verifying is performed periodically, and following a plurality of verification failures, the one or more user accounts are prevented from executing the deployed application (**8320**). In some embodiments, the verifying includes checking for multiple instances of the software application being simultaneously executed by the one or more user accounts (**8322**). In some embodiments, the system prevents access by the user to data stored at the one or more servers, upon determining that the one or more user accounts has been disabled (**8324**).

[0536] FIG. **84** is a flowchart representing a server method **8400** for syndicated deployment of a software application, according to certain embodiments of the invention. Server method **8400** may be governed by instructions, as described earlier.

[0537] At one or more marketplace servers hosting a marketplace application, in response to a request from a syndi-

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 212

cated server to distribute a software application from the marketplace, one or more user accounts associated with the request are identified (**8402**). The system verifies that the one or more user accounts has permission to use the software application (**8404**). The software application is deployed to the one or more user accounts, in accordance with license terms associated with the software application.

[0538] In some embodiments, the software application is presented for deployment to a user (**8408**). In some embodiments, the software application is made available for distribution through the syndicated server. In some embodiments, deploying includes providing through an application deployer, across a network to the one or more user accounts, a software application stored at the application repository (**8412**).

[0539] In some embodiments, deploying includes selecting one of a plurality of software applications, compatible with the one or more user accounts, from the application repository (**8414**). In some embodiments, the application repository stores software applications in a plurality of states, including at least one selected from the group consisting of a ready to deploy state, an undergoing quality assurance state, a ready to submit for quality assurance state, and an unfinished state (**8416**).

[0540] FIG. **85** is a flowchart representing a server method **8500** for licensing and receiving payment for a software application, according to certain embodiments of the invention. Server method **8500** may be governed by instructions, as described earlier.

[0541] At one or more servers hosting a marketplace application (e.g., FIGS. **129**, **155**), a software application is made available for distribution through the marketplace application (**8502**). A user request to license the software application is received (**8504**). The software application is provided for deployment to one or more user accounts, hosted on one or more servers (**8506**). In some embodiments, the software application is provided across a network for deployment at one or more servers associated with the user, hosting the one or more user accounts (**8508**).

[0542] In some embodiments, a selection by the user is processed that includes adding the software application to a cart (e.g., FIG. **160**, **172**, cart and checkout) associated with the user, and checking out the cart (**8510**). In some embodiments, prior to deploying, a description of license terms associated with the software application is presented (e.g., FIG. **158**, license) to the user (**8512**). In some embodiments, the license terms are specified by a vendor (e.g., FIG. **132**) associated with the software application (**8514**). In some embodiments, the license terms are selected from a plurality of options provided by a licensing engine associated with the one or more servers hosting a marketplace application (**8516**). In some embodiments, the system validates that the request to license the software application complies with the license terms (**8518**). In some embodiments, the request to license the software application includes a payment (e.g., FIG. **170**, payment info) selected from the group consisting of a cash payment, a credit payment, and a prospective future payment (**8520**).

[0543] In some embodiments, the software application is a web-based software application, executed at one or more servers (**8522**).

Application Framework

[0544] Some embodiments provide an application framework that facilitates synchronizing data between applications. The application framework can include data structures and a set of rules for synchronizing data between two applications. For example, consider a customer relationship management (CRM) application and an email application, both of which include tasks for a user. The fields (or columns, etc.) to be synchronized between the CRM application and the email application are first mapped to application framework data structure. For example, the task identifier fields for these two applications can be mapped to the "identifier" field in the application framework data structure. During a synchronization operation between the CRM application and the email application, the CRM application can first synchronize itself with the application framework data structure. The email application can then synchronize itself with the application framework data structure. Synchronization rules can be used to map the fields from the CRM application to the application framework data structure and to map the fields of the application framework data structure to fields for the email application. Note that that the synchronization process can be performed in both directions (e.g., from the CRM application to the email application and from the email application to the CRM application. Also note that in these embodiments, the code and/or the architecture of the applications do not need to be modified in order to synchronize data between the applications. In some embodiments, the data is synchronized between the databases for the applications without using explicit commands executed by the applications. In other words, the synchronization of data occurs at the database level, and not the application level.

[0545] FIG. **63** is a block diagram illustrating an exemplary application framework **6300**, according to some embodiments. In some embodiments, the application framework **6300** is located on one or more servers. For example, the application framework **6300** can be located on a server such as the servers **4420**-**4840** in FIG. **44**, any one of the servers **4520**-**4530** in FIG. **45**, the hosting infrastructure **4630** and/or the server **4602** in FIG. **46**, any one of the servers **4802** and/or **4830** in FIG. **48**, the server **4902** in FIG. **49**, the server **5002** in FIG. **50**, the server **5102** in FIG. **51**, any one of the server **5202** and/or the syndicated server **5230** in FIG. **52**, any one of the servers **5350**, **5450**, and **5550** in FIGS. **53**-**55**, respectively, and/or any one of the servers **5702**, **5754**, and **5764** in FIG. **57**. The one or more servers can include an end user server maintained by an end user (e.g., a customer-provisioned server, etc.) and a hosted server maintained by a service provider (e.g., a software vendor, a web hosting company, etc.). In some embodiments, the application framework **6300** is located on one or more client computer systems. For example, the application framework **6300** can be located on a client computer system such as the client systems **4450**-**4454** in FIG. **44**, and/or any one of the client systems **5820** and **5840** in FIG. **58**.

[0546] In some embodiments, the application framework **6300** includes one or more of an account management system (EMS) **6302** and/or a software stack **6340**. Recall that "EMS" is used to describe a generic account management system. The EMS **6302** can create and manage accounts within the application framework **6300**, provide user authentication, and/or synchronize applications within an account and/or between account frameworks. The EMS **6302** can include one or more of a database **6304** and/or an application synchronization module **6306**. The database **6304** is described in more detail with reference to FIGS. **64** and **72**-**73** below and the application synchronization module **6306** is described in

more detail below with reference to FIGS. **65-79** below. Note that although the discussion below refers to a single database, multiple databases can be used. The software stack **6340** can include one or more of: an email module **6342**, a web server module **6344**, an authoring module **6346**, a browser module **6348**, and/or auxiliary modules **6350**. The email module **6342** can include any electronic mail program. For example, the email module **6342** can include email servers (e.g., Sendmail, Postfix, and qmail, etc.), mail filtering programs (e.g., procmail, SpamAssassin, etc.), client email programs (e.g., elm, pine, Microsoft Outlook, etc.), etc. The web server module **6344** can include any application that can respond to client requests for pages, data, graphics, videos, documents, etc, hosted on a server coupled to clients and/or other servers via a network. For example, the web server module **6344** can include Apache HTTP Server, Microsoft Internet Information Server, etc. The authoring module **6346** can include any application that allows a developer or a user to generate web-based applications. For example, the authoring module **6346** can include a text editor, a word processor, a web development environment (e.g., Microsoft Expression Web, Adobe Dreamweaver, etc.), etc. The browser module **6348** can include any application with a rendering engine that can access data and/or services on a local and/or a remote computer system and render the results so that a user can view the data and/or interact with the services. In some embodiments, browser module **6348** is a web browser. The auxiliary modules **6350** can include other applications such as a Lightweight Data Access Protocol (LDAP) interface module and/or database, an Internet Message Access Protocol (IMAP) module, a Post Office Protocol (POP) module, a Dovecot module (e.g., an IMAP and POP server), a web server (e.g., Apache HTTP server, Microsoft IIS Server, etc.), one or more database management systems (e.g., MySQL, PostgreSQL, etc.), logging applications (e.g., Syslog-ng, etc.), authentication services (e.g., saslauthd, etc.). Note that although FIG. **63** illustrates a single module for each type of service, the software stack **6340** can include any number of modules for a given type of service. For example, two different email modules can be included within software stack **6340**.

[0547] In some embodiments, the application framework **6300** includes one or more accounts. For example, as illustrated in FIG. **63**, the application framework **6300** includes an account **6310**. The account **6310** can include one or more of: an account directory **6312**, and a database **6314**. The account directory **6312** can include a directory structure configured to store one or more applications. The account database **6314** can include meta data for the account **6310**. The meta data can include descriptions for the software packages **6316**, users associated with the account **6310**, and/or default synchronization hooks/rules (e.g., for user management, licensing, etc.).

[0548] In some embodiments, the account **6310** includes software packages **6316**. For example, the software packages **6316** can include utilities (e.g., phpmyadmin, phppgadmin, websvn, EDK, etc.), tools, and/or shared libraries (e.g., Zend, PHPunit, famfamfam, the Etelos Application Server Framework, etc.) used by applications within an account.

[0549] In some embodiments, the account **6310** includes one or more applications **6320-1** to **6320-N**. For example, the applications **6320** can include applications purchased through the marketplace **4410** in FIG. **44**. In some embodiments, one or more of the applications **6320** are web-based applications. In these embodiments, a user can interact with one or more of the applications **6320** using a browser engine (e.g., the browser module **6348**). In some embodiments, the applications **6320** are located on a remote server separate and distinct from a computer system and/or a server that includes the browser module **6348**. In these embodiments, the browser module **6348** can access the applications **6320** that are executed on the remote server via a network connection. In some embodiments, the applications **6320** are located on the same computer system that includes the browser module **6348**. In these embodiments, the browser module **6348** can access the applications **6320** that are executed on the local computer system. In some embodiments, the applications **6320** are located both on a remote server separate and the local computer system. In these embodiments, the browser module **6348** can access the applications **6320** that are executed on the remote server via a network connection. If a connection to the remote server is not available (e.g., the network connection is not available, the remote server is unavailable, etc.), the browser module **6348** can access the applications **6320** on the local computer system. These embodiments are discussed above in reference to the AOP mode of operation. In some embodiments, the browser module **6348** can access the applications **6320** on the local computer system and the remote server.

[0550] In some embodiments, a respective application **6320** includes one or more of: a database **6322**, files **6326**, and/or a version control repository **6328**. The databases **6322** can include data for the respective application **6320**. The files **6326** can include application files for the respective application **6320**. For example, the files **6326** can include static HTML files, scripts, graphics files, video files, XML files, access control files, documents, etc. The version control repository **6328** can include one or more versions of the files **6326**. For example, the version control repository **6328** can include repositories managed by version control systems such as Subversion (SVN), Concurrent Version System (CVS), Revision Control System (RCS), etc. In some embodiments, the files **6326** can be modified by developers through a version control interface (e.g., WebDAV). When making changes to the files **6326**, the version control system maintains a history of the changes, which can be used to generate patches, updates, and/or new versions of the files **6326** that can be distributed and deployed to other instances of the respective application (e.g., the application can be running on different account on a different server, etc.). In some embodiments, the files **6326** and the version control repository **6328** for a respective application are associated with one or more vhosts **6324**. A virtual host is a mechanism that allows a web server to host one or more domains. For example, www.example.com and www.example2.com can both be hosted on the same web server using a virtual host mechanism. Each virtual host may include a separate directory hierarchy within the web server and/or a separate database (e.g., either on the web server or on a separate database server or servers). A virtual host (e.g, vhost) for a respective application can point to a different set of files that are owned and/or used by the respective application.

[0551] In some embodiments, one or more of the applications **6320** are configured to use a specified email domain. In these embodiments, the email module **6342** can be configured by the applications **6320** to send and/or receive email on the specified email domain.

[0552] Note that a newly-created account may not include any applications. Also note that although FIG. **63** illustrates a

single account, the application framework **6300** can support any number of accounts (e.g., **0**, **1**, **2**, . . . , N). Moreover, although components are grouped together in FIG. **63**, these groupings can be changed. For example, the software stack **6340** can include all or a subset of the software packages **6316**.

[0553] FIG. **64**A is a block diagram illustrating exemplary components of an account **6420**, according to some embodiments. The account **6420** can be the account **6310** in FIG. **63**. The account **6420** can include one or more of an account directory **6422**, a database **6424**, software packages **6426**, and applications **6430** (including one or more of databases **6432**, files **6436**, version control repository **6438**, and vhost **6434**), which correspond in general to the account components described above with reference to FIG. **63**. In some embodiments, the account **6420** includes domains **6428**, which include one or more Internet domain names associated with applications within the account **6420**.

[0554] In some embodiments, an EMS database **6402** includes one or more of: a users table **6404**, an accounts table **6406**, a databases table **6408**, a domains table **6410**, an applications table **6412**, and/or a vhosts table **6414**. Recall that "EMS" is used to describe a generic account management system. The users table **6404** can include user records for one or more users that can be associated with one or more accounts. The accounts table **6406** can include account records that can include information about accounts, locations of account directories, users associated with the accounts, etc. The databases table **6408** can include "database" records that can include information about databases for the account **6420** (e.g., the database **6424**, the database **6432**, etc.). The domains table **6410** can include domain records for one or more domains associated with the account **6420** and/or the applications **6430**. The applications table **6412** can include application records that can include information about the applications **6430**, the files **6436** and directories within the account directory **6422**, the version control repositories **6438**, and/or the vhosts **6434** associated with the account **6420**. The vhosts table **6414** can include virtual host records that can include information associated with a vhost directory, an account domain, and/or the version control repository **6438**.

[0555] FIG. **64**A also includes a software stack **6450**, which can correspond to the software stack **6340** in FIG. **63**.

[0556] FIG. **64**B is a flow diagram of an exemplary process **6460** for creating an account and installing applications into the account, according to some embodiments. Note that the circled numbers in FIG. **64**B correspond to the circled numbers in FIG. **64**A. In some embodiments, an EMS (e.g., the EMS **6302**) performs these operations. The discussion of FIG. **64**B below describes the operations as being performed by the EMS for the sake of clarity. However, it should be noted that other modules can perform all or a subset of these operations. The process **6460** may be governed by instructions that are stored in a computer readable storage medium and that are executed by one or more processors of one or more servers. Each of the operations shown in FIG. **64**B may correspond to instructions stored in a computer memory or computer readable storage medium. The computer readable storage medium may include a magnetic or optical disk storage device, solid state storage devices such as Flash memory, or other non-volatile memory device or devices. The computer readable instructions stored on the computer readable storage medium are in source code, assembly language code, object code, or other instruction format that is interpreted by one or more processors.

[0557] The EMS creates an admin user record for an admin user in the admin users table **6404** (**6462**). In some embodiments, an application synchronization operation and/or an AOP synchronization operation cannot be performed for admin users. Note that non-admin users associated with an account can also be created in a separate process after the account has been created. These non-admin users can be stored in a "users" table (not shown). In some embodiments, an application synchronization operation and/or an AOP synchronization operation can be performed for non-admin users.

[0558] Returning to FIG. **64**B, the EMS creates an account record in the accounts table **6406** and an account directory that are associated with the admin user record (**6464**). The EMS then creates an account database and an associated "database" record in the databases table **6408** for the newly created account database (**6466**). Next, the EMS creates a domain record in the domains table **6410** (**6468**). The EMS then creates an application directory and an application record in the applications table **6412** associated with the application directory (**6470**). The EMS then creates an application database and a "database" record in the databases table **6408** associated with the application database (**6472**). Next, the EMS creates a vhost directory, a version control repository for the vhost directory, and/or a vhosts record in the vhosts table **6414** (**6474**). Note that a given application (e.g., the application **6430**) can have many virtual hosts (e.g., the vhost **6434**, etc.) associated with it. Thus, multiple vhost directories, version control repositories, and/or vhosts records can be associated with the given application. In some embodiments, a virtual host (e.g., the vhost **6434**) for an application (e.g., the application **6430**) can be associated with a database. In these embodiments, a database connection string, which indicates how the virtual host can connect to the database, can be stored in the databases table **6408** and/or the vhosts table **6414**. The EMS then downloads and installs the software packages **6426** (**6476**). The EMS then creates configuration files for the modules in the software stack **6450** (**6478**).

[0559] Although FIG. **64**B describes an exemplary process for creating a new account and installing applications into the new account, only a subset of the operations in FIG. **64**B may need to be performed. For example, if a user record for a given user and an account record associated with the user already exist in the EMS database **6402**, then steps **6462**-**6466** can be omitted when adding a new application to the existing account.

Synchronizing Applications

[0560] Some embodiments provide techniques for synchronizing data between applications. In these embodiments, the code and/or the architecture of the applications do not need to be modified in order to synchronize data between the applications. In some embodiments, the data is synchronized between the databases for the applications without using explicit commands executed by the applications. In other words, the synchronization of data occurs at the database level, and not the application level. In some embodiments, the application framework described above is used to facilitate synchronizing data between applications. For example, the data that can be synchronized between applications can

include tasks, contacts, calendar items, etc. The data is typically stored in tables within a database. The tables in which the data is stored can be different for each application. For example, a first application may store contact data in a "contacts" table in a database for the first application, whereas a second application may store contact data in an "email" table in a database for the second application. In some embodiments, one or more of the applications is a web-based application.

[0561] FIG. **65** is a block diagram **6500** of a server **6510** and a client **6530**, according to some embodiments. In some embodiments, the server **6510** includes an application framework **6512**. The application framework **6512** can include one or more accounts. For example, the application framework **6512** can include an account **6514**. The account **6514** can include one or more applications **6516**. The applications **6516** can be associated with application databases **6518**, which can store, update, delete, retrieve, query, and search for data in application databases **6518**. As illustrated in FIG. **65**, the account **6514** includes the applications **6516-1** and **6516-2** and the associated application databases **6518-1** and **6518-2**, respectively.

[0562] In some embodiments, a subset of the data stored in the application databases **6518-1** and **6518-2** can be synchronized with each other. For example, the subset of the data can be data that has changed (e.g., new data, updated data, deleted data, etc.). In some embodiments, the subset of the data stored in the application databases **6518-1** and **6518-2** are first synchronized with a data framework **6519** based on synchronization rules **6520**. The synchronization rules **6520** are then applied to the data that was synchronized to the data framework **6519** to synchronize this data with the respective application database. Alternatively, the synchronization rules can be applied directly to synchronize the subset of the data stored in the application databases **6518-1** and **6518-2** without using the data framework **6519**. The synchronization rules **6520** are described in more detail below. In some embodiments, the applications **6516-1** and **6516-2** can be applications that do not have any built-in capabilities to synchronize data with each other. These applications can include, but are not limited to, email applications, CRM applications, calendar applications, etc. If these applications are standalone applications, they may not be able to synchronize data with each other. For example, GOOGLE™ calendar may not be able to synchronize tasks with SUGAR CRM. In some embodiments, the application databases **6518-1** and **6518-2** are synchronized with each other directly. For example, a database that includes contact information for GOOGLE™ calendar can synchronize directly with a database that includes contact information for SUGAR CRM. This synchronization can be performed "outside" of the applications. In other words, the applications do not need to be executing or active to perform the synchronization.

[0563] In some embodiments, the client **6530** includes an application framework **6532**. The application framework **6532** can include one or more accounts. For example, the application framework **6532** can include an account **6534**. The account **6534** can include one or more applications **6536**. The applications **6536** can be associated with application databases **6538**, which can store, update, delete, retrieve, query, and search for data in application databases **6538**. As illustrated in FIG. **65**, the account **6534** includes the applications **6536-1** and **6536-2** and the associated application databases **6538-1** and **6538-2**, respectively. For example, the applications **6536** can be GOOGLE™ calendar and SUGAR CRM, respectively, and the databases **6538** can be databases for GOOGLE™ calendar and SUGAR CRM, respectively.

[0564] In some embodiments, a subset of the data stored in the application databases **6538-1** and **6538-2** can be synchronized with each other. For example, the subset of the data can be data that has changed (e.g., new data, updated data, deleted data, etc.). In some embodiments, the subset of the data stored in the application databases **6538-1** and **6538-2** are first synchronized with a data framework **6539** based on synchronization rules **6540**. The synchronization rules **6540** are then applied to the data that was synchronized to the data framework **6539** to synchronize this data with the respective application database. For example, if changes have occurred in GOOGLE™ calendar, fields (or columns, etc.) in the database for GOOGLE™ calendar first synchronized with the data framework **6539**, then the synchronization rules **6540** are applied to the data framework **6539** to propagate/synchronize the data from the data framework **6539** to the database for SUGAR CRM. Alternatively, the synchronization rules can be applied directly to synchronize the subset of the data stored in the application databases **6538-1** and **6538-2** without using the data framework **6539**. The synchronization rules **6530** are described in more detail below. In some embodiments, the applications **6536-1** and **6536-2** can be applications that do not have any built-in capabilities to synchronize data with each other. In some embodiments, the application databases **6538-1** and **6538-2** are synchronized with each other directly.

[0565] In some embodiments, the applications **6516** in the account **6514** on the server **6510** can be synchronized with the applications **6536** in the account **6534** on the client **6530**. In some embodiments, the data frameworks **6519** and **6539** are first synchronized with each other, and then the synchronization rules **6520** and **6540**, respectively, are applied to the synchronized data to synchronize the application databases **6518** and **6538**, respectively. In some embodiments, the AOP synchronization process described above is used to synchronize the application databases **6518** on the server **6510** with the application databases **6538** on the client **6530**.

[0566] Note that although FIG. **65** illustrates a single account in the server **6510** and a single account in the client **6530**, more than one account can be included on the server **6510** and/or the client **6530**. Furthermore, each account can include any number of applications.

[0567] In some embodiments, the data frameworks **6519** and **6539** and the sync rules **6520** and **6540** are included in an application synchronization module (e.g., the application synchronization module **6306** in FIG. **63**.

[0568] Although FIG. **65** illustrates that both the server **6510** and the client **6530** include application frameworks, in some embodiments, either the server **6510** or the client **6530** may not include an application framework.

[0569] FIG. **66** is a flow diagram of an exemplary process **6600** for synchronizing applications, according to some embodiments. Note that the circled numbers in FIG. **66** correspond to the circled numbers in FIG. **65**. The process **6600** may be governed by instructions that are stored in a computer readable storage medium and that are executed by one or more processors of one or more servers. Each of the operations shown in FIG. **66** may correspond to instructions stored in a computer memory or computer readable storage medium. The computer readable storage medium may include a magnetic or optical disk storage device, solid state storage devices such as Flash memory, or other non-volatile memory device

or devices. The computer readable instructions stored on the computer readable storage medium are in source code, assembly language code, object code, or other instruction format that is interpreted by one or more processors.

[0570] As illustrated in FIG. **66**, changes are detected in an application database (e.g., the application database **6518-1** in FIG. **65**) for a first application (e.g., the application **6516-1**) when a user uses the first application (**6602**). The changes are then pushed to a data framework (e.g., the data framework **6519** in FIG. **65**) (**6604**) and the changes are then synchronized with a second application (e.g., the application **6516-2** in FIG. **65**) (**6606**). The second application has synchronized the changes (**6614**). In some embodiments, the changes are synchronized between the server (e.g., the server **6510** in FIG. **65**) and the client (e.g., the client **6530** in FIG. **65**) (**6616**). In these embodiments, the AOP synchronization technique described above can be used.

[0571] In some embodiments, synchronizing the changes with the second application includes pulling changes from the data framework (**6608**), applying synchronization rules to the changes (**6610**), and applying the changes to an application database (e.g., the application database **6518-2** in FIG. **65**) for the second application (**6612**).

[0572] Note that the process **6600** can be performed at the client **6530** and/or the server **6510** in FIG. **65**. Moreover, changes from any applications within an application framework can be synchronized with other applications within the application framework. For example, changes made in the application **6516-2** can be synchronized with the application **6516-1** and changes made in the application **6516-1** can be synchronized with the application **6516-2**. Furthermore, changes made in applications on a server/client can be synchronized with applications on a client/server (e.g., using the AOP synchronization technique described above).

[0573] FIG. **67** is a block diagram illustrating an exemplary process **6700** for synchronizing applications, according to some embodiments. The process **6700** may be governed by instructions that are stored in a computer readable storage medium and that are executed by one or more processors of one or more servers. Each of the operations shown in FIG. **67** may correspond to instructions stored in a computer memory or computer readable storage medium. The computer readable storage medium may include a magnetic or optical disk storage device, solid state storage devices such as Flash memory, or other non-volatile memory device or devices. The computer readable instructions stored on the computer readable storage medium are in source code, assembly language code, object code, or other instruction format that is interpreted by one or more processors. In some embodiments, the process is performed on a computer system that is to be synchronized. For example, the computer system can be a server or a client computer system.

[0574] An outbound process module **6702** receives an outbound order **6740**, which includes changes to data for one or more applications and an order that changes to data are applied. Note that the changes to the data can be made to one or more databases, one or more structured files, and/or one or more documents. In some embodiments, the ordering of changes to data in the outbound order **6740** can be: all updates **6741**, all inserts **6742**, all deletes **6743**, parent deletes **6744**, child deletes **6745**, parent inserts **6746**, and child inserts **6747**. The all updates **6741** includes all updates to existing data. The all inserts **6742** includes all new data records that are to be added. The all deletes **6743** includes all existing data records that are to be deleted. The parent deletes **6744** include parent data records that are to be deleted. The child deletes **6745** include child data records that are to be deleted. The parent inserts **6746** include parent data records that are to be deleted. The child inserts **6747** include child data records that are to be deleted.

[0575] In some embodiments, an outbound execute module **6704** receives the changes to the data for one or more applications and executes one or more of: an outbound insert module **6710** that inserts new data records, an outbound update module **6720** that updates existing data records, and an outbound delete module **6730** that deletes existing data records for one or more applications **6750**. The one or more applications **6750** can also receive changes to the data from an AOP synchronization process through the user_in: Hooks module **6772**. The user_in:Hooks module **6772** can include an auto increment module **6774**, which is described in more detail above with reference to FIGS. **27-29** above.

[0576] In some embodiments, the outbound insert process **6710** includes hooks **6711** and filters **6715**. The hooks **6711** can include one or more of: hooks to a backdata update module **6712** that performs an initial data synchronization (e.g., as described in FIGS. **18** and **19** above), hooks to a licensing module **6713** that verifies that licenses are valid (e.g., as described in FIGS. **44** and **57** above), and/or hooks to a user management module **6714** (e.g., as described in FIG. **57** above). Note that the term "hooks" refers to a mechanism by which a given module can access another module. For example, a hook can include a function, a procedure, and/or a method within a given module that can access functions of another module, an API function, procedure, and/or method, etc. In some embodiments, the backdata update module **6712** requeues new data records (e.g., data records to be added) for an application as updates that are updated in the outbound update process **6720**. The filters **6715** can include an owner id translation filter **6716** that translates owner_ids between applications. Note that the term "filter" refers to a mechanism that can perform a specified operation on data. For example, this operation can include a translation operation that translates data values, a selection operation that selects data values based on specified criteria, etc.

[0577] In some embodiments, the outbound update process **6720** includes hooks **6721** and filters **6724**. The hooks **6721** can include one or more of: hooks to a licensing module **6713** that verifies that licenses are valid and/or hooks to a user management module **6714**. The filters **6722** can include an owner_id translation filter **6716** that translates owner_ids between applications.

[0578] In some embodiments, the outbound delete process **6730** includes hooks **6731**. The hooks **6731** can include one or more of: hooks to a licensing module **6713** that verifies that licenses are valid and/or hooks to a user management module **6714**.

[0579] Note that if the licensing modules **6713** determine that a license for an application is not valid, the outbound insert process **6710**, the outbound update process **6720**, and/or the outbound delete process **6730**, respectively, are not performed. Similarly, if the user management modules **6714** determine that a user of an application does not have sufficient privileges to change specified data for the one or more applications, the outbound insert process **6710**, the outbound update process **6720**, and/or the outbound delete process **6730**, respectively, are not performed.

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 217

[0580] In some embodiments, as changes to the data for the one or more applications **6750** are detected by an inbound module **6752**, the changes can be accumulated for one or more of an AOP synchronization process and an application synchronization process. If the changes are being accumulated for an application synchronization process (**6754**, App Sync), the changes are accumulated into an account synchronization log **6762**, which is then used by an outbound process **6764**. If the changes are being accumulated for an AOP synchronization process (**6754**, AOP), the changes are accumulated into an outgoing synchronization log **6756**. Changes from the outgoing_sync_log and the AOP user_in are then separated into "user out" tables **6758** as described above with reference to FIG. **27** above. The user_out tables are then received by an AOP sync_in process **6760** on a receiving system (e.g., a server if the changes were made on a client, or a client if the changes were made on the server).

[0581] In some embodiments, an AOP sync_out process **6766** receives changes to data for applications on a transmitting system made when the applications were operating in an AOP mode (e.g., as described above). These changes can be included in a user_in table **6768**. The changes included in the user_in tables are then processed **6770**. In some embodiments, the changes included in the user_in tables are received by the outbound process **6764**, which incorporates the changes made to applications within a given computer system (e.g., application synchronization) and changes made to applications on different computer systems (e.g., AOP synchronization). In some embodiments, the changes included in the user_in tables are processed by the auto increment module **6774** as described above. After being processed by the auto increment module **6774**, the changes included in the user_in tables are incorporated into the user_out tables **6758** and/or the one or more applications **6750**. Note that the changes included in the user_in tables can be included in the user_out tables because an application synchronization operation may include changes both to the instance of the application on the server and on the client.

[0582] FIG. **68** is a block diagram illustrating an exemplary process **6800** for handling parent-child relationships during a synchronization process, according to some embodiments. The exemplary process **6800** describes exemplary operations performed by the owner_id translation module **6716** in FIG. **67**. Parent-child relationships can be unenforced or can be enforced using foreign keys. Parent-child relationships can be used to associate records with each other. These associations can include a one-to-one (e.g., one parent to one child), a one-to-many (e.g., one parent to multiple children), and a many-to-many (e.g., many parents to many children) relationship. As illustrated in FIG. **68**, applications **6802** are being synchronized with each other. The applications **6802** can be any of the applications described above (e.g., web-based applications). The applications **6802** include users tables **6804**, apply_users tables **6806**, applications tables **6808**, and databases tables **6810**. A user with a given id is associated with an application with an associated id via the apply_users table **6806**. For example, the apply_user table **6806-1** can include foreign keys referring to a data record in the users table **6804-1** associated with a given user (e.g., user_id) and a data record in the applications table associated with a given application (e.g., app_id). The apply_user table **6806-1** can also include a unique record id (e.g., "id") and an owner_id value that corresponds to the given user's instance within an application. The corresponding set of tables also exist for the application **6802-2**.

[0583] In some embodiments, the parent-child relationships are defined in sync_vmap_parent tables **6812** and sync_vmap_child tables **6814**. The sync vmap_parent tables **6812** include an app_table field, an app_column field, a database_id field, a translate field, an auto_inc field, and a primary key field. The app_table and app_column fields specify an application table and column, respectively. The database_id field specifies the database_id in the databases table **6408** corresponding to the database including the app_table and app_column. The translate field specifies whether an owner_id translation is to be performed for a given parent-child relationship defined in the sync_vmap_parent tables **6812**. The auto_inc field specifies that the parent value is an auto incrementing value (e.g. used in AOP sync). The primary key field indicates that the parent value is a primary key of a table (e.g., used for AOP synchronization operations and application synchronization operations). For application synchronization operations, the primary key can indicate whether the EMS needs to generate a unique identifier for a given field. For AOP synchronization operations, the primary keys are required to be the same on the server and the client. Thus, the primary key can be used to reinforce the parent/child relationships. The sync_vmap_child tables **6814** include the app_table field, an app_column field, and a parent_id field.

[0584] In some embodiments, during a synchronization process, the changes to the data are stored one or more sync log entries. An exemplary sync log entry **6816** is illustrated in FIG. **68**. The exemplary sync log entry **6816** includes one or more of: sync_id (e.g., a unique identifier for the sync log entry), transaction (e.g., a unique identifier for a given database transaction), account_sync (e.g., the type of synchronization data—AOP sync, application sync, etc.), sync_column (e.g., the column to be synchronized), broadcast_table (e.g., a table in the source database), broadcast_column (e.g., a column in a table that in the source database includes the changed data), broadcast_database_id (e.g., a database_id in the databases table **6408** corresponding to the source database), broadcast_action (e.g., an action performed by the source database—insert, update, delete, etc.), broadcast_vmap_id (e.g., a unique identifier for a source vmap), datatypes (e.g, the data type of the changed data), new_bool (e.g., if a column is a Boolean value and an Insert or an Update has occurred, this stores the new Boolean value), old_bool (e.g., if a column is a Boolean value and an Update has occurred, this stores the old Boolean value), new_text (e.g., if a column is not a Boolean value and an Insert or an Update has occurred, this stores the new value), old_text (e.g., if a column is not a Boolean value and an Update has occurred, this stores the old value), group_ids (e.g., identifies ownership and record filtering for AOP and/or application synchronization operations when group ownership is declared), map_id (e.g., a unique identifier that is used to identify records across applications), owner_id (e.g., an identifier for the user that corresponds to the user's instance with an application), backdata (e.g., whether a backdata update is to be performed), from_aop (e.g., whether the sync log entry resulted from an AOP synchronization operation), subscribe_action (e.g., an action to be performed by the subscribing database—insert, update, delete, etc.), subscribe_database_id (e.g., a database_id in the databases table **6408** corresponding to the subscribing database), subscribe_table (e.g., a table in the subscribing database), subscribe_column (e.g., a column in a table that in

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 218

the subscribing database includes the changed data), deploy_licensing_id (e.g., a unique identifier used to identify the licensing model for a given application), vmap_ids (e.g., a unique identifier for a destination/subscribing vmap), and special (e.g., used to mark phrases—updating, complete, etc.—for auto-increment).

[0585] In some embodiments, every table in the application used by the synchronizer includes a column "eas_sync_map_id." During a synchronization operation, the field "map_id" includes the unique identifier that corresponds to the value in eas_sync_map_id column. This column can include a 32 character unique identifier which is used to identify records across applications. For example, when updating or deleting a value during synchronization, the value in the eas_sync_map_id column is used to match to the eas_sync_map_id in the destination app.

[0586] In some embodiments, the sync log entry **6816** includes data that allows the lookup of parent/children relationships. This lookup can be performed while the sync log entry is being processed. Note that the fields prefixed with "broadcast_" include data pertaining to the source database and the fields prefixed with "subscribe_" include data pertaining to the destination database.

[0587] In some embodiments, if a subscribing database includes a parent/child relationship defined on a subscribe_table and/or a subscribe_column, the sync log entry is treated as a special case and is handled in a different order. (e.g., see the outbound order **6740** in FIG. **67**). In some embodiments, a subscribing database is a database that has requested to receive changes to data made in another database. In some embodiments, if a given sync log entry corresponds to a data record that has a parent-child relationship and the data record is marked as "translate" (e.g., via the sync_vmap_parent **6812** tables), the apply_user records for the source database are queried to determine a user associated with the owner_id for the changed data that the sync log entry is referencing. The apply_user records for the subscribing database are also queried to determine the owner_id for the subscribing database associated with the determined user. The owner_id value of the subscribing database can then be used to replace the value of the owner_id in the sync log entry.

[0588] FIG. **69** is a flow diagram of an exemplary process **6900** for translating owner IDs, according to some embodiments. The process **6900** may be governed by instructions that are stored in a computer readable storage medium and that are executed by one or more processors of one or more servers. Each of the operations shown in FIG. **69** may correspond to instructions stored in a computer memory or computer readable storage medium. The computer readable storage medium may include a magnetic or optical disk storage device, solid state storage devices such as Flash memory, or other non-volatile memory device or devices. The computer readable instructions stored on the computer readable storage medium are in source code, assembly language code, object code, or other instruction format that is interpreted by one or more processors. For example, the process **6900** can be performed by the application synchronization module **6306** (e.g., in a client and/or a server) in FIG. **63**. The discussion of FIG. **69** below describes the operations as being performed by the application synchronization module **6306** of a computer system (e.g., a client and/or a server) for the sake of clarity. However, it should be noted that other modules can perform all or a subset of these operations.

[0589] In FIG. **69**, the application synchronization module **6306** determines whether the app column for a subscribing database is a vmap_parent or a vmap_child of a parent that has the "translate" field set to "true" (**6902**). If the "translate" field is set to "false" (**6904**, no), the application synchronization module **6306** continues with the synchronization (**6910**). If the "translate" field is set to "true" (**6904**, yes), the application synchronization module **6306** obtains the owner_id corresponding to the user of the subscribed database from the apply_user table (**6906**). Next, the application synchronization module **6306** replaces the app_column value with the owner_id from the apply_user table (**6908**). The application synchronization module **6306** then continues with the synchronization (**6910**).

[0590] FIG. **70** is a block diagram illustrating an exemplary process **7000** for merging users between applications, according to some embodiments. FIG. **71** is a flow diagram of an exemplary process **7100** for merging users between applications, according to some embodiments. It is desirable to merge users between applications because a single user may have different user identifiers (e.g., user_ids) in different applications. For example, the same user may have a user identifier of 2 in a first application and a user identifier of 6 for a second application. It is therefore desirable to merge these two users into a single user so that the synchronization of user data can be facilitated. The process **7100** corresponds to the process **7000** and are discussed together. Note that the circled numbers in FIG. **70** correspond to the circled numbers in FIG. **71**. The process **7100** may be governed by instructions that are stored in a computer readable storage medium and that are executed by one or more processors of one or more servers. Each of the operations shown in FIG. **71** may correspond to instructions stored in a computer memory or computer readable storage medium. The computer readable storage medium may include a magnetic or optical disk storage device, solid state storage devices such as Flash memory, or other non-volatile memory device or devices. The computer readable instructions stored on the computer readable storage medium are in source code, assembly language code, object code, or other instruction format that is interpreted by one or more processors.

[0591] FIG. **70** includes applications **7004-1** and **7004-2** and EMS **7008**, which can be included in an application framework (e.g., the application framework **6300**) on a computer system (not shown). A user is created in the applications **7004-1** and **7004-2** for a user **7002** (**7102**). For example, the user can have a user_id of 4 in the application **7004-1** and a user_id of 3 in the application **7004-2**. A user is also created in the EMS **7008** (**7104**). For example, the user can have a user_id of 6 in the EMS **7008**. In some embodiments, the EMS user_ids are applied to the applications **7004-1** and **7004-1** (**7106**) and the owner_id values are set (**7108**). Note that applying users to applications identifies the user's relationship and access to the application. The user's relationship is the "owner_id," which is the user's unique identifier for the application. Also note that a user cannot access an application using AOP unless the user is applied to that application and the user is allowed to use AOP (e.g., the AOP flag set for the user). Furthermore, applications may use this record to authenticate a user for an application. Thus, the application can query the EMS to determine whether the user has permission to use the application.

[0592] Returning to FIGS. **70** and **71**, the user_ids are merged in the EMS (**7110**). Thus, user_id=4 for the applica-

tion **7004-1** and user_id=3 for the application **7004-2** correspond to the same user_id=6 in the EMS **7008**.

[0593] At a later time, the user **7002** creates a task **7006** associated with the user_id=3 in the application **7004-2** (**7112**). Note that although the task **7006** is described in this example, other objects can be created (e.g., calendar items, contacts, documents, etc.). Next, the task **7006** is synchronized with the EMS **7008** (**7114**). In doing so, the EMS **7008** determines the EMS user_id corresponding to the user_id=3 for the application **7004-2** (e.g., user_id=6). The EMS **7008** then translates the user id in the task **7006** from user_id=3 for the application **7004-2** to user_id=4 for the application **7004-1** (**7116**). The EMS **7008** then sends the task **7006** with the translated user_id to the application **7004-1** (**7118**). The application **7004-1** receives the task **7006** with the translated user_id and creates a task in the database for the application **7004-1** corresponding to the task **7006** generated by the application **7004-2** (**7120**).

[0594] FIG. **72** presents a block diagram of an exemplary server **7200**, according to some embodiments. The server **7200** generally includes one or more processing units (CPU's) **7202**, one or more network or other communications interfaces **7204**, memory **7210**, and one or more communication buses **7208** for interconnecting these components. The communication buses **7208** may include circuitry (sometimes called a chipset) that interconnects and controls communications between system components. The server **7200** may optionally include a display **7206** and one or more input devices **7205** (e.g., keyboard, mouse, trackpoint, etc.). The memory **7210** includes high-speed random access memory, such as DRAM, SRAM, DDR RAM or other random access solid state memory devices; and may include non-volatile memory, such as one or more magnetic disk storage devices, optical disk storage devices, flash memory devices, or other non-volatile solid state storage devices. The memory **7210** may optionally include one or more storage devices remotely located from the CPU(s) **7202**. The memory **7210**, or alternately the non-volatile memory device(s) within the memory **7210**, comprises a computer readable storage medium. In some embodiments, the memory **7210** stores the following programs, modules and data structures, or a subset thereof: operating system **7211**, network communication module **7212**, an EMS module **7213**, an account framework **7230**, a software stack **7250**, and/or auxiliary modules **7260**.

[0595] The operating system **7211** includes procedures for handling various basic system services and for performing hardware dependent tasks. Network communication module **7212** can be used for connecting the server **7200** to other computers via the one or more communication network interfaces **7204** (wired or wireless) and one or more communication networks, such as the Internet, other wide area networks, local area networks, metropolitan area networks, and so on.

[0596] In some embodiments, the EMS module **7213** includes one or more of: an EMS database **7214** and/or an application synchronization module **7215**. The application synchronization module can include one or more of: a data framework **7216**, synchronization rules **7218**, synchronization logs **7219**, filters **7220**, and hooks **7221**. Note that these components and/or modules are described in more detail with reference to FIGS. **63-71**, and **74-79**. In some embodiments the data framework **7216** includes a synchronization data structure **7217** that can be used to synchronize data between applications. The synchronization data structure **7217** is described in more detail with reference to FIGS. **74-79**.

[0597] In some embodiments, the account framework **7230** includes one or more of: an account directory **7231**, an account database **7232**, software packages **7233**, and/or one or more applications **7240**. The applications **7240** can include one or more of: an application database **7241**, files **7243**, and version control repository **7244**. The files **7243** and the version control repository **7244** can be associated with a vhost **7242**. In some embodiments, the software packages **7233** can include utilities (e.g., phpmyadmin, phppgadmin, websvn, EDK, etc.), tools, and/or shared libraries (e.g., Zend, PHPunit, famfamfam, the Etelos Application Server Framework, etc.) used by applications within an account. These components and/or modules are described in more detail with reference to FIGS. **63-71**, and **74-79**.

[0598] In some embodiments, the software stack **7250** includes one or more of: an email module **7251**, a web server module **7252**, an authoring module **7253**, and/or other services **7254**. Email module **7251** can include any electronic mail program. For example, the email module **7251** can include email servers (e.g., Sendmail, Postfix, and qmail, etc.), mail filtering programs (e.g., procmail, SpamAssassin, etc.), client email programs (e.g., elm, pine, Microsoft Outlook, etc.), etc. The web server module **7252** can include any application that can respond to client requests for pages, data, graphics, videos, documents, etc, hosted on a server coupled to clients and/or other servers via a network. For example, the web server module **7252** can include Apache HTTP Server, Microsoft Internet Information Server, etc. The authoring module **7253** can include any application that allows a developer or a user to generate web-based applications. For example, the authoring module **7253** can include a text editor, a word processor, a web development environment (e.g., Microsoft Expression Web, Adobe Dreamweaver, etc.), etc. The other services **7254** can include other applications such as a Lightweight Data Access Protocol (LDAP) interface module and/or database, an Internet Message Access Protocol (IMAP) module, a Post Office Protocol (POP) module, a Dovecot module (e.g., an IMAP and POP server), a web server (e.g., Apache HTTP server, Microsoft IIS Server, etc.), one or more database management systems (e.g., MySQL, PostgreSQL, etc.), logging applications (e.g., Syslog-ng, etc.), authentication services (e.g., saslauthd, etc.). Note that although FIG. **72** illustrates a single module for each type of service, the software stack **7250** can include any number of modules for a given type of service. For example, two different email modules can be included within software stack **7250**.

[0599] In some embodiments, the auxiliary services module **7260** includes one or more of: famfamfam (icon images), phpMyAdmin (php web-based MySQL database management interface), phpPGAdmin (php web-based PostgSQL database management interface), WebSVN (php web-based Subversion interface), ImageMagick (image manipulation library), ZendFramework (php utility framework), IconCube Loaders (encrypted-php decryption library), libpng (PNG manipulation library), libjpeg (JPEG manipulation library), Neon (WebDAV client library), mcrypt (encryption library), and FreeType (font utilities library).

[0600] Each of the above identified elements may be stored in one or more of the previously mentioned memory devices, and corresponds to a set of instructions for performing a function described above. The above identified modules or programs (i.e., sets of instructions) need not be implemented as separate software programs, procedures or modules, and

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 220

thus various subsets of these modules may be combined or otherwise re-arranged in various embodiments. In some embodiments, the memory **7210** may store a subset of the modules and data structures identified above. Furthermore, the memory **7210** may store additional modules and data structures not described above.

[0601] Although FIG. **72** shows a "server," FIG. **72** is intended more as functional description of the various features that may be present in a set of servers than as a structural schematic of the embodiments described herein. In practice, and as recognized by those of ordinary skill in the art, items shown separately could be combined and some items could be separated. For example, some items shown separately in FIG. **72** could be implemented on single servers and single items could be implemented by one or more servers. The actual number of servers used to implement an application server and how features are allocated among them will vary from one implementation to another, and may depend in part on the amount of data traffic that the system must handle during peak usage periods as well as during average usage periods.

[0602] FIG. **73** presents a block diagram of an exemplary client computer system **7300**, according to some embodiments. The client computer system **7300** generally includes one or more processing units (CPU's) **7302**, one or more network or other communications interfaces **7304**, memory **7310**, and one or more communication buses **7308** for interconnecting these components. The communication buses **7308** may include circuitry (sometimes called a chipset) that interconnects and controls communications between system components. The client computer system **7300** includes a display **7306** and one or more input devices **7305** (e.g., keyboard, mouse, trackpoint, etc.). The memory **7310** includes high-speed random access memory, such as DRAM, SRAM, DDR RAM or other random access solid state memory devices; and may include non-volatile memory, such as one or more magnetic disk storage devices, optical disk storage devices, flash memory devices, or other non-volatile solid state storage devices. The memory **7310** may optionally include one or more storage devices remotely located from the CPU(s) **7302**. The memory **7310**, or alternately the non-volatile memory device(s) within the memory **7310**, comprises a computer readable storage medium. In some embodiments, the memory **7310** stores the following programs, modules and data structures, or a subset thereof: operating system **7311**, network communication module **7312**, an EMS module **7313**, an account framework **7330**, a software stack **7350**, a browser module **7356**, and/or auxiliary modules **7360**.

[0603] The operating system **7311** includes procedures for handling various basic system services and for performing hardware dependent tasks. Network communication module **7312** can be used for connecting the server **7300** to other computers via the one or more communication network interfaces **7304** (wired or wireless) and one or more communication networks, such as the Internet, other wide area networks, local area networks, metropolitan area networks, and so on.

[0604] In some embodiments, the EMS module **7313** includes one or more of: an EMS database **7314** and/or an application synchronization module **7315**. The application synchronization module can include one or more of: a data framework **7316**, synchronization rules **7318**, synchronization logs **7319**, filters **7320**, and hooks **7321**. Note that these components and/or modules are described in more detail with reference to FIGS. **63-71** and **74-79**. In some embodiments the data framework **7316** includes a synchronization data structure **7317** that can be used to synchronize data between applications. The synchronization data structure **7317** is described in more detail with reference to FIGS. **74-79**.

[0605] In some embodiments, the account framework **7330** includes one or more of: an account directory **7331**, an account database **7332**, software packages **7333**, and/or one or more applications **7340**. The applications **7340** can include one or more of: an application database **7341**, files **7343**, and version control repository **7344**. The files **7343** and the version control repository **7344** can be associated with a vhost **7342**. In some embodiments, the software packages **7233** can include utilities (e.g., phpmyadmin, phppgadmin, websvn, EDK, etc.), tools, and/or shared libraries (e.g., Zend, PHPunit, famfamfam, the Etelos Application Server Framework, etc.) used by applications within an account. These components and/or modules are described in more detail with reference to FIGS. **63-71**, and **74-79**.

[0606] In some embodiments, the software stack **7350** includes one or more of: an email module **7351**, a web server module **7352**, an authoring module **7353**, and/or other services **7354**. Email module **7351** can include any electronic mail program. For example, the email module **7351** can include email servers (e.g., Sendmail, Postfix, and qmail, etc.), mail filtering programs (e.g., procmail, SpamAssassin, etc.), client email programs (e.g., elm, pine, Microsoft Outlook, etc.), etc. The web server module **7352** can include any application that can respond to client requests for pages, data, graphics, videos, documents, etc, hosted on a server coupled to clients and/or other servers via a network. For example, the web server module **7352** can include Apache HTTP Server, Microsoft Internet Information Server, etc. The authoring module **7353** can include any application that allows a developer or a user to generate web-based applications. For example, the authoring module **7353** can include a text editor, a word processor, a web development environment (e.g., Microsoft Expression Web, Adobe Dreamweaver, etc.), etc. The other services **7354** can include other applications such as a Lightweight Data Access Protocol (LDAP) interface module and/or database, an Internet Message Access Protocol (IMAP) module, a Post Office Protocol (POP) module, a Dovecot module (e.g., an IMAP and POP server), a web server (e.g., Apache HTTP server, Microsoft IIS Server, etc.), one or more database management systems (e.g., MySQL, PostgreSQL, etc.), logging applications (e.g., Syslog-ng, etc.), authentication services (e.g., saslauthd, etc.). Note that although FIG. **73** illustrates a single module for each type of service, the software stack **7350** can include any number of modules for a given type of service. For example, two different email modules can be included within software stack **7350**.

[0607] In some embodiments, the auxiliary services module **7360** includes one or more of: famfamfam (icon images), phpMyAdmin (php web-based MySQL database management interface), phpPGAdmin (php web-based PostgSQL database management interface), WebSVN (php web-based Subversion interface), ImageMagick (image manipulation library), ZendFramework (php utility framework), IconCube Loaders (encrypted-php decryption library), libpng (PNG manipulation library), libjpeg (JPEG manipulation library), Neon (WebDAV client library), mcrypt (encryption library), and FreeType (font utilities library).

[0608] In some embodiments, the browser module **7356** can include any application with a rendering engine that can

access data and/or services on a local and/or a remote computer system and render the results so that a user can view the data and/or interact with the services. In some embodiments, browser module **7356** is a web browser.

[0609] Each of the above identified elements may be stored in one or more of the previously mentioned memory devices, and corresponds to a set of instructions for performing a function described above. The above identified modules or programs (i.e., sets of instructions) need not be implemented as separate software programs, procedures or modules, and thus various subsets of these modules may be combined or otherwise re-arranged in various embodiments. In some embodiments, the memory **7310** may store a subset of the modules and data structures identified above. Furthermore, the memory **7310** may store additional modules and data structures not described above.

[0610] FIG. **74** presents a block diagram illustrating an exemplary application synchronization module **7400**, according to some embodiments. The application synchronization module **7400** can be the application synchronization module **6304** in FIG. **63**. In some embodiments, the cross-application synchronization module **7400** includes one or more of: synchronization rules **7402**, sync log **7410**, data framework **7420**, filters **7430**, hooks **7440**, an outbound insert module **7450**, an outbound update module **7451**, an outbound delete module **7452**, an outbound process module **7453**, an outbound execute module **7454**, an outbound order **7455**, an inbound module **7456**, and/or an AOP module **7457**. These components and/or modules are described in more detail with reference to FIGS. **1-42** and **63-79**.

[0611] In some embodiments, the synchronization rules **7402** include synchronization rules for one or more applications **7404**. In some embodiments, the data framework **7420** includes a synchronization data structure **7422**. The synchronization data structure **7422** is described in more detail with reference to FIG. **75**. In some embodiments, the filters **7430** include an owner_id translation module **7431** (e.g., as described in FIG. **67**). In some embodiments, the hooks **7440** include one or more of: a hook to a back data update module **7440**, a hook to a licensing module **7441**, a hook to a user management module **7442**, and a hook to an auto increment module **7443** (e.g., as described in FIG. **67**).

[0612] FIG. **75** presents exemplary synchronization data structures **7500**, according to some embodiments. The synchronization data structures **7500** include one or more of: an eas_sync_app_log data structure **7502**, an inbound data structure **7504**, and an outbound data structure **7506**. The eas_sync_app_log **7502** is used to store the changes made by the application. These changes can be tracked using database triggers on the tables and/or columns to be synchronized. The inbound data structure **7504** includes data that is to be processed by the application. This inbound data can be first placed in a temporary table prior to being processed. The outbound data structure **7506** includes data that is to be prepossed and grouped together for outbound processing. This processed data can them be placed in an outgoing_sync_log and/or an account_sync_log. If the data is in an account_sync_log, the data is to be outbound processed and sent to the application databases. If the data is in the outgoing sync_log, the data is to be sent to the user_out and/or the sync_out (for AOP) table to wait for an AOP sync. The data can be placed into the user_in or sync_in (for AOP) and processed as if it was in the account_sync_log.

[0613] FIG. **76** is a flow diagram of an exemplary process **7600** for synchronizing applications, according to some embodiments. The process **7600** may be governed by instructions that are stored in a computer readable storage medium and that are executed by one or more processors of one or more servers. Each of the operations shown in FIG. **76** may correspond to instructions stored in a computer memory or computer readable storage medium. The computer readable storage medium may include a magnetic or optical disk storage device, solid state storage devices such as Flash memory, or other non-volatile memory device or devices. The computer readable instructions stored on the computer readable storage medium are in source code, assembly language code, object code, or other instruction format that is interpreted by one or more processors.

[0614] As FIG. **76** illustrates, changes made to a first data set in a plurality of data sets is detected (**7602**). At least a first subset of the changes are synchronized to a data framework (e.g., the data framework **7420**) that facilitates data synchronization between the plurality of data sets (**7204**). In some embodiments, a data set and a data framework includes one or more of: one or more databases, one or more structured files, and one or more documents. The structured file can include one of: an extensible markup language (XML) file, a hypertext markup language (HTML) file, and a file including values separated by a delimiter. The delimiter can include one of: a comma, and a tab break. A document can include one of: a web document, a word processor document, a spreadsheet document, a presentation document, and an electronic mail message. In some embodiments, a change to a data set includes one or more of: an update of data in the data set, an addition of data to the data set, and/or a deletion of data from the data set.

[0615] In some embodiments, synchronizing the at least the first subset of the changes to the data framework includes determining a mapping between data fields in a data structure for the first data set and data fields in a data structure for the data framework (**7606**) and synchronizing the data fields in the data structure for the first data set corresponding to the at least the first subset of the changes with the data fields in the data structure for the data framework based on the mapping (**7608**). In some embodiments, synchronizing the data fields includes translating values for the data fields in the data structure for the first data set to corresponding values for the data fields in the data structure for the data framework based on translation rules for the first data set (**7610**). In some embodiments, the translation rules and/or the mapping are included in the synchronization rules (e.g., the synchronization rules **6520**, **6540** and **7218** in FIGS. **65** and **72**, respectively).

[0616] In some embodiments, at least a second subset of the synchronized changes are synchronized from the data framework to a second data set in the plurality of data sets (**7612**). In some embodiments, synchronizing the at least the second subset of the synchronized changes from the data framework to a second data set in the plurality of data sets includes: determining a mapping between data fields in a data structure for the second data set and data fields in a data structure for the data framework (**7614**) and synchronizing the data fields in the data structure for the second data set with the data fields in the data structure for the data framework based on the mapping (**7616**). In some embodiments, synchronizing the data fields includes translating values for the data fields in the data framework to corresponding values for the data fields in the

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 222

second data set based on translation rules for the second data (**7618**). In some embodiments, the translation rules and/or the mapping are included in the synchronization rules (e.g., the synchronization rules **6520**, **6540** and **7218** in FIGS. **65** and **72**, respectively).

[0617] In some embodiments, at least a third subset of the synchronized changes are synchronized from the data framework to a second data framework (**7620**).

[0618] FIG. **77** is a flow diagram of an exemplary process **7700** for generating synchronization rules that are used to synchronize applications, according to some embodiments. The process **7700** may be governed by instructions that are stored in a computer readable storage medium and that are executed by one or more processors of one or more servers. Each of the operations shown in FIG. **77** may correspond to instructions stored in a computer memory or computer readable storage medium. The computer readable storage medium may include a magnetic or optical disk storage device, solid state storage devices such as Flash memory, or other non-volatile memory device or devices. The computer readable instructions stored on the computer readable storage medium are in source code, assembly language code, object code, or other instruction format that is interpreted by one or more processors.

[0619] As illustrated in FIG. **77**, a first data set in a plurality of data sets that is to be synchronized with a data framework is identified (**7702**). A mapping between one or more data fields in a data structure for the first data set and one or more data fields in a data structure for the data framework is determined (**7704**). Synchronization rules for the first data set based on the determined mapping are generated (**7706**).

[0620] In some embodiments, at least a subset of the one or more data fields for the first data set that include data values that are to be translated to corresponding data values for the data fields for the data framework are identified (**7708**). Translation rules for translating data values for the at least the subset of the data fields for the first data set to corresponding data values for the data fields for the data framework are generated (**7710**). In some embodiments, the translation rules are included in the synchronization rules (e.g., the synchronization rules **6520**, **6540** and **7218** in FIGS. **65** and **72**, respectively).

[0621] In some embodiments, the first data set and the data framework include one or more of: one or more databases; one or more structured files; and one or more documents. In some embodiments, the first data set and the data framework include one or more databases. The mapping can include a mapping between columns of one or more database tables for the first data set and columns of one or more database tables for the data framework.

[0622] FIG. **78** is a flow diagram of an exemplary process **7800** for synchronizing applications, according to some embodiments. The process **7800** may be governed by instructions that are stored in a computer readable storage medium and that are executed by one or more processors of one or more servers. Each of the operations shown in FIG. **78** may correspond to instructions stored in a computer memory or computer readable storage medium. The computer readable storage medium may include a magnetic or optical disk storage device, solid state storage devices such as Flash memory, or other non-volatile memory device or devices. The computer readable instructions stored on the computer readable storage medium are in source code, assembly language code, object code, or other instruction for mat that is interpreted by one or more processors.

[0623] As illustrated in FIG. **78**, changes made to a first data set for a first web-based application in an account are detected (**7802**). At least a second data set for a second web-based application in the account is identified (**7804**), wherein the second data set includes at least a subset of the data included in the first data set. Synchronization rules to synchronize the second data set with the first data set are applied (**7806**).

[0624] In some embodiments, the synchronization rules include: one or more rules to synchronize at least a subset of data fields in a data structure for the first data set with data fields in a data structure for a data framework, and/or one or more rules to synchronize at least a subset of the data fields in the data structure for the data framework with data fields in a data structure for the second data set.

[0625] In some embodiments, the account includes a plurality of data sets and corresponding web-based applications.

[0626] In some embodiments, the synchronization rules include one or more of: a mapping between fields in the first data set and fields in the second data set and/or rules for resolving data conflicts between the first data set and the second data set.

[0627] FIG. **79** presents exemplary synchronization rules data structures **7900**, according to some embodiments. The synchronization rules data structures include a sync_vmap data structure **7902**, a sync_vmap_parent data structure **7904**, and a sync_vmap_child data structure **7906**.

[0628] Screenshots

[0629] In the following figures (descriptions of screenshots), all references to modules (e.g., marketplace module **6122**, user account access module **6172**, deployer module **6152**, packager module **6136**, and marketplace database management module **6180**) relate to modules shown in the system of FIG. **61**. These screen shots are exemplary screenshots of user interfaces that display information for, and enable user interaction with, the various applications and components of the application marketplace of the present invention.

[0630] FIG. **101** is an exemplary screenshot **10100** of an Account User Management Interface, generated by User Account Access Module **6172**. The information reflected here is stored in a user database accessed by User Account Database Management Module.

[0631] An Account User Management Interface allows a user to activate services and application for different users. Merging users together between applications enables better synchronization of data between and across applications. Field **10102** displays a user's first name; field **10104** displays a user's last name; field **10106** displays a user's phone number; field **10108** displays a user's email address; field **10110** displays a user's office; field **10112** displays a user's department; field **10114** is a matrix of user access information. These are exemplary fields; in other embodiments different fields may be created.

[0632] FIG. **102** is an exemplary screenshot **10200** of an application packager, generated by Packager Module **6136**. The information reflected here is stored in a user database accessed by User Account Database Management Module **6188**.

[0633] The Application Packager allows a user to package up the user's application for distribution in the Marketplace or move the application to another hosted solution (e.g. an Ete-

los hosted solution). Field **10202**, and **10204** allow a user to input the user's Marketplace Username, and user's Marketplace password respectively.

[0634] FIG. **103** is an exemplary screenshot **10300** of an application packager, generated by Packager Module **6136**. The information entered here is stored in a database accessed by Deployment Database Management Module **6184**. Package a new full app (with the associated Begin button) **10302** allow a user to begin packaging a new application. Package an update to an app **10304**, allows a user to package an update to an application.

[0635] FIG. **104** is an exemplary screenshot **10400** of an application packager, generated by Packager Module. The information entered here is stored in a database accessed by Deployment Database Management Module **6184**. Field **10402** allows a user to enter an application name which the user wishes to package. Field **10404** allows a user to specify a script associated with the package.

[0636] FIG. **105** is an exemplary screenshot **10500** of an application packager, generated by Packager Module **6136**. The information entered here is stored in a database accessed by Deployment Database Management Module **6184**. Package a new full app (with the associated Begin button) **10502** allow a user to begin packaging a new application. Package an update to an app, **10504**, allows a user to package an update to an application.

[0637] FIG. **106** is an exemplary screenshot **10600** of an application packager, generated by Packager Module **6136**. The information entered here is stored in a database accessed by Deployment Database Management Module **6184**. Field **10602** allows a user to enter an application name which the user wishes to package. Field **10604** allows a user to specify other data.

[0638] FIG. **107** is an exemplary screenshot **10700** of an application packager, generated by Packager Module **6136**. The information entered here is used by Permissions Module **6128** to verify the user's credentials for a selected destination store (e.g. an Etelos store).

[0639] FIG. **108** is an exemplary screenshot **10800** of an application packager, generated by Packager Module **6136**. Field **10802** displays a set of icons associated with packaging of a software application for distribution.

[0640] FIG. **109** is an exemplary screenshot **10900** of an application packager, generated by Packager Module **6136**. Field **10902** shows a file structure for a packaged application.

[0641] FIG. **110** is an exemplary screenshot **11000** of an application packager, generated by Packager Module **6136**. Field **11002** allows a user to view Binders and Tables associated with a packaged application. Field **11004** allows a user to select a specific binder and/or table.

[0642] FIG. **111** is an exemplary screenshot **11100** of an application packager, generated by Packager Module **6136** for specifying a script associated with a packaged application.

[0643] FIG. **112** is an exemplary screenshot **11200** of an application packager, generated by Packager Module **6136**. Scheduler, **11202**, allows a user to schedule software applications for packaging updates. Database Broadcast/Subscribe rules, **11204**, allows an administrator to specify broadcasting and subscription rules for updates based on an application's license as stored in licensing database management module **6182**.

[0644] FIG. **113** is an exemplary screenshot **11300** of an application packager, generated by Packager Module **6136**.

[0645] FIG. **114** is an exemplary screenshot **11400** of an Account Information Details, generated by User Accounts Module **6156**.

[0646] Account Information Details shows the user the detailed information associated with each application installed on the user's account. Field **11402** includes examples of information that may be included in one embodiment.

[0647] FIG. **115** is an exemplary screenshot **11500** of a Development Environment that can be used to create or modify applications. Files panel **11502** displays the file structure associated with an application. Binders panel **11504** displays Binders associated with the application. Panel **11506** can display the contents of selected files, folders, and/or objects.

[0648] FIG. **116** is an exemplary screenshot **11600** of an application packager, generated by Packager Module **6136** similar to FIG. **103**.

[0649] FIG. **117** is an exemplary screenshot **11700** of an AOP sync rules manager, generated by an application synchronization module (e.g., the application synchronization module **6306** in FIG. **63**). The AOP sync rules manager **11702** displays current synchronization rules as defined by a user. Column **11704** displays the Code associated with an application. The code column includes the code to execute upon rule execution. Column **11706** displays the status of a rule associated with an application. Column **11708** displays names of applications and/or database for which there is a sync rule defined. Column **11710** and **11712** display a table and a column, respectively, in the application database for which there is a defined sync rule. Button **11714** opens an interface to specify whether a given column in the application database is an autoincrementing column. Button **11716** allows a user to create a new sync rule.

[0650] FIGS. **118-121** are exemplary screenshots of an AOP sync rules manager. FIG. **118** is an exemplary screenshot **11800** of an AOP sync rules manager, allowing a user to define a new sync rule. Field **11802** allows a user to specify the application and/or database for which the user wishes to define a new sync rule. Field **11804** allows the user to choose a table associated with the application and/or database for which the user wishes to define a sync rule.

[0651] FIG. **119** is an exemplary screenshot **11900** of an AOP sync rules manager, allowing a user to define a new sync rule. Field **11902** allows a user to specify an application and/or a database for which the user wishes to define a new sync rule. Field **11904** allows the user to choose a table associated with the application and/or database for which the user wishes to define a new sync rule. Field **11906** allows the user to specify a Column associated with the application and/or database for which the user is defining a new sync rule. Fields **11908**, **11910**, and **11912** allow a user to define a table in the application framework (e.g., Framework Sync Table) to which the table defined in field **11904** is synchronized, an associated column (e.g., Framework Sync Column) in the table defined in field **11904** to which the column defined in field **11906** is synchronized, and a scope of the synchronization rule, respectively.

[0652] FIG. **120** is an exemplary screenshot **12000** of an AOP sync rules manager, allowing a user to define a new sync rule. Field **12002** allows a user to specify an application and/or a database for which the user wishes to define a new sync rule. Field **12004** allows the user to choose a table associated with the application and/or database for which the

user wishes to define a new sync rule. Field **12006** allows the user to specify a Column associated with the application and/or database for which the user is defining a new sync rule. Field **12008** allows a user to select when and what types of changes are synchronized. For example, changes can be synchronized when there are one or more of the following: inserts (e.g., new data added), and updates (e.g., existing data is updated), deletes (e.g., existing data is deleted), etc. Fields **12010** and **12012** allow a user to define a table in the application framework (e.g., Framework Sync Table) to which the table defined in field **12004** is synchronized, and a scope of the synchronization rule, respectively.

[0653] FIG. **121** is an exemplary screenshot **12100** of an AOP sync rules manager, allowing a user to define a new sync rule. Field **12102** allows a user to specify an application and/or a database for which the user wishes to define a new sync rule. Field **12104** allows the user to choose a table associated with the application and/or database for which the user wishes to define a new sync rule. Field **12106** allows the user to specify a Column associated with the application and/or database for which the user is defining a new sync rule. Field **12108** allows a user to select when and what types of changes are synchronized. Fields **12110**, **12112**, and **12114** allow a user to define a table in the application framework (e.g., Framework Sync Table) to which the table defined in field **12104** is synchronized, an associated column (e.g., Framework Sync Column) in the table defined in field **12104** to which the column defined in field **12106** is synchronized, and a scope of the synchronization rule, respectively.

[0654] FIG. **122** is an exemplary screenshot **12200** of a Group Manager, generated by an access module (e.g., the access module **6170** in FIG. **61**). In some embodiments, a user may define an application and/or database **12202**, group **12204**, membership **12206**, membership id **12208**, data type **12210**, group primary key **12212**, membership group foreign key **12214**, and Data id **12216** associated with an application. By clicking the Create button **12218** a user may create a group definition associated with an application. By clicking the Cancel button **12220**, a user may cancel the group definition associated with an application.

[0655] FIG. **123** is an exemplary screenshot **12300** of a sync rules manager wherein a user generates an ID Translation Definition for parent-child relationships. The user may define a Vmap Parent **12302** by defining a table **12304** and a column **12306** for an application and/or a database that is a parent for another column in a table. The user may save changes by clicking the Save button **12308**, or cancel changes by clicking the Cancel button **12310**.

[0656] FIG. **124** is an exemplary screenshot **12400** of a sync rules manager wherein a user generates an ID Translation Definition. Field **12402** includes applications for which there is an ID Translation Definition. Field **12404** can display the Vmap parent relationships described in FIG. **123**. The button Edit **12406** allows a user to edit the relationships information associated with an application.

[0657] FIG. **125** is an exemplary screenshot **12500** of a sync rules manager wherein a user may use Task Editor **12502** to add scripts to a sync rule. The user may specify whether the task is to be performed before or after syncing using field **12504**. The user may also specify the type of scripts using the field **12506** (e.g., SQL). The user may further specify in field **12508** whether the script is to evaluate or to execute.

[0658] FIG. **126** is an exemplary screenshot **12600** of a sync rules manager wherein a user may use Task Editor **12602** to add script details to a sync rule similar to FIG. **125**.

[0659] FIG. **127** is an exemplary screenshot **12700** of a sync rules manager wherein a user can manage synchronization rules associated with an application. The user may access help information by clicking the question mark button **12702**. The user may view a scope of a rule in field **12704**, a code in field **12706**, a status of a rule in field **12708**, when a rule is applied in field **12710**, an application and/or a database in field **12712**, an application table in field **12714**, an application column in filed **12716**, an account table in field **12718**, application account in field **12720**. The user may specify an ID Translation by clicking on the plus button **12722**, or create a new sync rule by clicking on the plus button **12724**.

[0660] FIG. **128** is an exemplary screenshot **12800** of a sync rules manager wherein a user can manage synchronization rules associated with an application similar to FIG. **127**.

[0661] FIG. **129** is an exemplary screenshot **12900** of a support page (e.g. an Etelos Support page) wherein a user may use a Storefront **12902** (e.g. Etelos "My Storefront") to add product or services to a marketplace (e.g. an Etelos marketplace) generated by marketplace module **6122**. Field **12904** displays a store listing; field **12906** displays a status of an application; field **12908** displays a type of an application.

[0662] FIG. **130** is an exemplary screenshot **13000** of a Product/Service Basic Information Page generated by listing module **6124**. A user may specify a product name in field **13002**, short description in field **13004**, product type in field **13006**, and item status in field **13008**. Store Item Details appears in the field **13010**. The user may create a Product/Service Basic Information page by clicking on the button Create A Copy **13012**.

[0663] FIG. **131** is an exemplary screenshot **13100** of a staging application page. In application file field **13102**, a user can specify an application file for e.g. the VBS registration website manager (in one embodiment, VBS is an example software application listed in the marketplace). In application size field **13104**, a user can specify the application size. In install time field **13106**, a user can specify the estimated install time for the application.

[0664] FIG. **132** is an exemplary screenshot **13200** of an edit licensing page wherein a user may edit licensing for "a registration website manager" **13202**. The edit licensing for "registration website" is generated by the licensing module **6126**. The information entered on this page is stored by licensing database management module **6182**.

[0665] Field **13204** displays the name, description, or billing, retail, wholesale, and type (per account or per user) information associated with a packaged application for which the user may wish to edit licensing information. In field **13206**, the user may select an install type for an application. In field **13208**, the user may type a description of the application. In field **13210**, the user may specify a license version. In field **13212**, the user may specify a type of license associated with the application. In field **13214**, the user may specify a quantity of licenses associated with the application (allowance of application features like user groups, max records on quantifiable items like tasks, documents, projects, sales reps, etc. . . .). In field **13216**, the user may specify whether the license is a trial version. In field **13218**, the user may specify a billing type associated with a license associated with the application. In field **13220**, the user may specify whether the application is hosted using bundled hosting. In field **13222**,

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 225

the user may specify a hosting provider associated with the application. In field **13224**, the user may specify a retail pricing associated with the application. In field **13226**, the user may specify a resale term to be associated with a license associated with the application. In field **13228**, the user may specify a wholesale pricing associated with the application. In field **13230**, the user may specify a license type associated with a source code associated with the application (e.g. whether source code is licensed as an open source license, etc.). In field **13232**, the user may specify whether the source code associated with the application is downloadable. In field **13234**, the user may specify whether application synchronization (syncing) is enabled. In field **13236**, the user may specify whether the application is Application on Plane (AOP) enabled. In field **13238**, the user may specify whether multiple instances of the application are allowed. In field **13240**, the user may specify an external product ID associated with the application. In field **13242**, the user may specify whether there is an ad server associated with the application. In field **13244**, the user may specify the status of a license associated with the application (e.g. whether the license is on or off). These fields are exemplary, other embodiments may incorporate different fields and editing options concerning licensing associated with an application.

[0666] FIG. **133** is an exemplary screenshot **13300** of a setup marketing information page generated by the listing module **6124**. A product review and store categories page **13302** includes information used to display an application in a catalog and search result. In field **13304**, a user may upload a product image to be displayed with an application. In field **13306**, a user may specify a desired turnkey rating associated with an application. In field **13308**, a user may specify an internet address associated with a provider of an application. In field **13310**, a user may specify different categories associated with an application. In field **13312**, a user may specify store items related to an application. In field **13314**, a user may specify keywords for which an application may appear in a search result. In field **13316**, a user may specify a type of hosting associated with an application.

[0667] FIG. **134** is an exemplary screenshot **13400** of a setup marketing full page generated by the listing module **6124** whereby a user may describe his applications. In field **13402**, a user may specify a web address to redirect a purchaser to the user's marketing material. In field **13404**, a user may write a description of an application which the user is selling through marketplace module **6122**.

[0668] FIG. **135** is an exemplary screenshot of a setup product features full page **13500** generated by the listing module **6124**. In field **13502**, a user may turn on WYSIWYG. In field **13504**, a user may write a description of features in the user's application. The user's application is available for sale through marketplace module **6122**.

[0669] FIG. **136** is an exemplary screenshot of a product demo page **13600** generated by the listing module **6124**. The information entered here is stored by marketplace database management module **6180**. A user may access this product demos page **13602** to upload screenshots, and videos associated with the user's applications available through Marketplace module **6122**. In field **13604**, a user may add a descriptive text describing the features of the user's application. In field **13606**, a user may upload videos to be used as a demo associated with an application. In field **13608**, a user may upload screenshots to be used as a demo associated with an application. In field **13610**, a user may view previously uploaded screenshots associated with an application.

[0670] FIG. **137** is an exemplary screenshot **13700** of an "Add a Blog" feed page, executed by marketplace module **6122**. A user can set up a blog feed for a store item using add a Blog feed page **13702** (e.g. "add a Blog feed for Etelos CRM"). In field **13704**, a user may provide a feed link. In field **13708**, a user may specify a feed status (e.g. on or off). In field **13708**, a user may specify whether blogs should be displayed on the user's store. Using buttons **13710**, **13712**, and **13714**, a user may save changes, update feed, or review feed respectively.

[0671] FIG. **138** is an exemplary screenshot **13800** of a setup frequently asked questions (FAQs) page generated by the listing module **6124**. Information entered here is stored in the Marketplace database management module **6180**. In field **13802**, a user may enter FAQs about the user's application. Using the checkbox **13804**, a user can display the FAQ tab on the user's store in Marketplace module **6122**.

[0672] FIG. **139** is an exemplary screenshot **13900** of a setup getting started information page generated by the listing module **6124**. The information entered here is stored by marketplace database management module **6180**. In field **13902**, an administrator may write a getting started page associated with an application which a user will see when starting the application. In field **13904**, an administrator may write a getting started email associated with an application which a user will receive when the application is installed. In field **13906**, an administrator may write a text associated with an application. The text would only appear if a user has purchased a trial version of the application.

[0673] FIG. **140** is an exemplary screenshot **14000** of a support information page generated by marketplace database management module **6180**. In field **14002**, an administrator may write a support page to be displayed to the licensees of the administrator's application in Marketplace module **6122**.

[0674] FIG. **141** is an exemplary screenshot **14100** of an about us page generated by the listing module **6124**. The information entered here is stored by marketplace database management module **6180**. Icon **14102**, Cart, enables a user to access items in the user's shopping cart. Icon **14104**, Account, enables a user to access the users account information. In field **14106**, an administrator may specify the administrator's company name. In field **14108**, an administrator may specify the administrator's company website link. In field **14110**, an administrator can specify a contact email associated with the administrator's application. In field **14112**, an administrator may write an "about us" section to be included with the administrator's application by Marketplace module **6122**.

[0675] FIG. **142** is an exemplary screenshot **14200** of a product upgrade page generated in some embodiments by the listing module **6124**, or in some embodiments by the licensing module. The information entered here is stored by marketplace database management module **6180**. Icon **14202**, Cart, enables an administrator to access items in the user's shopping cart. Cart icon **14204**, and account icon **15204** operate as previously described. Using checkbox **14206**, an administrator may specify whether an upgrade tab should be displayed in the administrator's store by marketplace module **6122**. In field **14208**, an administrator can enter a name for an upgrade. In field **14210**, an administrator can write a description of an upgrade. In field **14212**, an administrator may specify upgrade licensing information, e.g., the version or

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 226

features of a license to upgrade to. In field **14214**, an administrator may specify the status of an upgrade (e.g. if the upgrade is active). Using button **14216**, an administrator may create the upgrade. Field **14218**, displays upgrade information for an application an administrator is currently viewing.

[0676] FIG. **143** is an exemplary screenshot **14300** of a "My Store Style" page generated by the listing module **6124**. The information entered here is stored by marketplace database management module **6180**. An administrator can use the "My store style" page to setup the look and feel of his regular and/or syndicated store pages. For example, in one embodiment, in field **14302**, an administrator can specify page background color, page headings font color, body font color, and general font. In field **14304**, an administrator may write or define a page header to be displayed in the administrator's store. In field **14306**, an administrator may write or define a page footer to be displayed in the administrator's store in Marketplace module **6122**.

[0677] FIG. **144** is an exemplary screenshot **14400** of a pricing grid and purchase/license link setup page generated by the listing module **6124**. The information entered here is stored by marketplace database management module **6180**. Pricing grid and purchase/license link setup page **14402** enables an administrator to set up items that will appear in the pricing grid on the Marketing page for a product in Marketplace module **6122**. In field **14404**, an administrator may enter the name of an application for which the administrator is setting up a pricing grid. In field **14406**, an administrator may enter a price associated with an application. In field **14408**, an administrator may provide a web address to redirect a prospective purchaser once the purchaser clicks a buy now link. In field **14410**, an administrator may choose the link text. In field **14412**, the administrator may enter sorting order information. In field **14414**, the administrator may specify a status of a pricing grid (e.g. active). Using the create button **14416**, an administrator can create the pricing grid the administrator has specified.

[0678] FIG. **145** is an exemplary screenshot **14500** of a web services "Post Data" setup page generated by the listing module **6124**. The information entered here is stored by Marketplace database management module **6180**. In field **14502**, an administrator may specify a web address for a purchaser's purchase/license information to be posted to. Using radio buttons **14504**, an administrator may specify in what format he prefers to receive a user's purchase/license information. Using the save changes button **14506**, an administrator may save the changes the administrator has created. In field **14508**, an administrator may provide a sample of the format in which he prefers to receive a purchase/license information.

[0679] FIG. **146** is an exemplary screenshot **14600** of a support page (e.g. an Etelos support page) wherein a user may use a Storefront (e.g. Etelos "My Storefront") to add product or services to a marketplace (e.g. an Etelos marketplace) generated by marketplace module **6122**. Field **14602** displays a store listing and link; field **14604** displays a status of an application; field **14606** displays a type of an application; field **14608** displays tools available to a user. For example a user administrator can preview a store listing, number of people who have purchased from the store, and other information related to an application.

[0680] FIG. **147** is an exemplary screenshot **14700** of a support page (e.g. an Etelos support page) generated by marketplace module **6122**. Using drop down menu **14702**, a user may select which application the user would like to deploy to the user's customers. Using drop down menu **14704**, a user may select which licenses the user would like to upgrade. Using drop down menu **14706**, a user may specify a server filter. Using the begin upgrade button **14708**, a user may start an upgrade process.

[0681] FIG. **148** is an exemplary screenshot **14800** of a support page (e.g. an Etelos support page) generated by marketplace module **6122**. In field **14802**, a user may add new domains to the user's account. A user may view an account name in field **14804**, its associated default domain name in field **14806**, and its status in field **14808**. A user may add a domain to an account by clicking on the button Add Domain **14810**.

[0682] FIG. **149** is an exemplary screenshot **14900** of a support page (e.g. an Etelos support page) generated by marketplace module **6122**. A user may use the page "create a sub domain for" **14902** to create a subdomain associated with an account. In field **14904**, a user may type a domain the user would like to associate with the user's account.

[0683] FIG. **150** is an exemplary screenshot **15000** of a store product list generated by the marketplace module **6122**. The information reflected here is stored in marketplace database management module **6180**, including product name column **15002**, store owner column **15004**, company column **15006**, status column **15008**, and export option **15010**.

[0684] FIG. **151** is an exemplary screenshot **15100** of a store listing report (e.g. a store listing **15110**, "Store listing report for Etelos CRM") generated by transactions report module **6132**. In one embodiment, the store listing report **15110** utilizes a table including the following rows and columns. Column **15102** displays a running total for every row; column **15104** displays a total year to date for every row; columns in the field **15106** displays a total for a month; column **15108** displays a total for financial quarter 1. Rows **15112** display various totals for store listings. Row **15114** displays accounts that are later cancelled or deactivated. Rows in **15116** provide a break down between unused, dormant, dry accounts, and active accounts. Rows in **15118** provide a number of registered users and accounts registered without a registered user. Rows in **15120** provide a number of billable users, free users, and a total number of users. These columns and rows are exemplary, and in other embodiments different information might be provided in a store listing table.

[0685] FIG. **152** is an exemplary screenshot **15200** of a store listing report (e.g. a store listing **15110**, "Store listing report for Etelos CRM") generated by marketplace module **6122** similar to FIG. **151**.

[0686] FIG. **153** is an exemplary screenshot **15300** of a licensing transaction report (also known as a bottleneck report) generated by transactions report module **6132**. This report includes all non-hosting services, non-cancelled, non-automated billing items. In one embodiment, the report includes the following information: field **15302**, displays when an item was created, field **15304** displays the name-email, and company associated with an item; field **15306** displays a check mark if an item is store registered; field **15308** displays a check mark if an item is associated with a credit card; field **15310** displays an item's billing detail and associated domain; field **15312** displays a check mark if an item is checked out; field **15314** displays a check mark if an item is processed; field **15316** displays a check mark and a time stamp to indicate whether licensing terms associated with an item are accepted; field **15318** displays a check mark

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 227

to indicate whether an item is downloaded; field **15320** displays a check mark if an installation for an item has started; field **15322** displays a check mark to indicate whether an item is licensed; field **15324** displays a check mark to indicate whether an installation of an item is complete; and field **15326** displays a check mark to indicate whether a user associated with an item is registered.

[0687] FIG. **154** is an exemplary screenshot **15400** of a support page (e.g. an Etelos support page) generated by Marketplace module **6122** wherein a vendor or administrator may edit an application information using the field **15402**. Using the field **15404**, the vendor may view or select the vendor's packaged applications available for editing. Field **15406** displays the names of a vendor's applications available for editing. Field **15408** indicates whether an application has been deployed in Marketplace module **6122**. Field **15410** indicates an application type; field **15412** indicates a date where an application was created; field **15414** indicates the grouping information associated with an application; field **15416** indicates an application's version; field **15418** enables a user to archive an application using a check box.

[0688] FIG. **155** is an exemplary screenshot of a Marketplace **15500** (e.g. an Etelos Marketplace) generated by Marketplace module **6122**. Tab **15502** displays the available storefronts stored by Marketplace database management module **6180**. Tab **15504** displays the categories of available applications. Tab **15506** displays the most recent available applications. Tab **15508** displays a list of featured applications. Box **15510** indicates the steps a user takes to install and use the user's applications in a Marketplace.

[0689] FIG. **156** is an exemplary screenshot **15600** of a hosting and development environment (e.g. an Etelos Hosting and Dev Environment for BT Web21C SDK **15602**) wherein a user may purchase/license an application available through Marketplace module **6122**. Tab **15604** displays a purchasing information **15620** including purchasing options; tab **15606** displays available features of an application; tab **15608** displays available demos associated with an application; tab **15610** displays FAQs associated with an application; tab **15612** displays available forums associated with an application; tab **15614** displays a support page associated with an application. A user may purchase/license an application by clicking the Buy Now button **15616**. A user may obtain a free trial of an application by clicking on a Free Trial button **15618**.

[0690] FIG. **157** is an exemplary screenshot of a Marketplace **15700** (e.g. an Etelos Marketplace) generated by Marketplace module **6122**. **15702** and **15704** are examples of storefronts available through Marketplace database management module **6180**.

[0691] FIG. **158** is an exemplary screenshot **15800** of a licensing page. Field **15802** includes licensing terms which a user may accept by clicking the Accept button **15804**, or decline by clicking the decline button **15806**.

[0692] FIG. **159** is an exemplary screenshot **15900** of a support page (e.g. an Etelos support page) generated by Marketplace module **6122** including a user's billing **15902** and transaction **15904** history, including date **15906**, order ID **15908**, name **15910**, description **15912**, and price **15914**.

[0693] FIG. **160** is an exemplary screenshot **16000** of a shopping cart page. Field **16002** displays a date when a purchase/license is occurring; field **16003** provides a description; field **16004** displays a quantity of applications a user is purchasing; field **16006** displays a total cost of items in a user's shopping cart.

[0694] FIG. **161** is an exemplary screenshot **16100** of a Getting Started page for a CRM application listing (e.g. an Etelos CRM test listing page) where a user can initialize an application.

[0695] FIG. **162** is an exemplary screenshot **16200** of a Getting Started page for a developer toolkit app (e.g. an Etelos V6 developer toolkit page) where a user can initialize a development environment generated by listing module **6124** to define an application.

[0696] FIG. **163** is an exemplary screenshot **16300** of a support page (e.g. an Etelos support page) generated by marketplace module **6122**. Field **16302** describes an exemplary user idea called "Syndicated project management app". This description is stored by the marketplace database management module **6180**.

[0697] FIG. **164** is an exemplary screen shot **16400** from the bottom portion of the screenshot **16300** in FIG. **163**. In field **16402**, Add a Comment, a user may add a comment to a user idea reflected in FIG. **163**. To add a comment a user enters the user's email address in the field **16404**, and the user's name in the field **16406**. The user may include a web address in the field **16408**. The user can type a comment in the field **16410**. The user can add a comment by clicking on the button Add A Comment **16412**. These comments are stored by the marketplace database management module **6180**.

[0698] FIG. **165** is an exemplary screenshot **16500** of a support page (e.g. an Etelos support page) generated by marketplace module **6122**. Using button **16502**, a user may start a new idea. In field **16504**, a user may view most popular ideas as voted by other users.

[0699] FIG. **166** is an exemplary screenshot **16600** of an installation page generated by application deployer module **1650**. Installation in progress page **16602** displays which applications are being installed, estimated install times, status, and progress for each application installation. Field **16604** displays a name of an application being installed; field **16606** displays an estimated time for an application installation; field **16608** displays an installation status; field **16610** displays a progress bar associated with an application installation.

[0700] FIG. **167** is an exemplary screenshot **16700** of an installation processing page generated by application deployment module **6150**. A progress bar **16702** displays a progress of a user's request for installation. This page is displayed to a user during installation of a software application.

[0701] FIG. **168** is an exemplary screenshot **16800** of an installation processing page similar to FIG. **166**. This page shows that the installation has completed (**16806**) and gives a link to get started (**16808**) with the deployed application.

[0702] FIG. **169** is an exemplary screenshot **16900** of a support page (e.g. an Etelos support page) generated by marketplace module **6122**. This page is presented to a user when configuring accounts and applications. My developer accounts **16902** enables a user to select an account name to edit its properties, or add an account manually. Drop down menu **16904** enables a user to choose the type of account to generate. Button "Build an App" **16906** enables a user to build an application for a particular account in the list of developer accounts **16902**. Button "Add Domain" **16908** enables a user to add a domain associated with an application.

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 228

[0703] FIG. **170** is an exemplary screenshot **17000** of a support page (e.g. an Etelos support page) generated by marketplace module **6122**. This page is displayed to a user when configuring their account to run software apps. Payment information **17002** displays an account payment information. In one embodiment payment information field **17004** includes the last four digits of a credit card number associated with an account.

[0704] FIG. **171** is an exemplary screenshot **17100** of a marketplace page, generated by marketplace module **6122**. This page is presented to a user searching for software applications in the marketplace. Marketplace search result **17102** displays the results for an exemplary search. Support forum search results **17104** provides a link to search results in Marketplace forum stored in Marketplace database management module **6180**. Knowledge base search result **17106** provides a link to search results in Marketplace knowledge base stored in Marketplace database management module **6180**.

[0705] FIG. **172** is an exemplary screenshot **17200** of a marketplace shopping cart page, generated by marketplace module **6122**, similar to FIG. **141**. This screen shows a shopping cart with a software application for licensing, displayed to a prospective licensee.

[0706] FIG. **173** is an exemplary screenshot **17300** of a support page (e.g. an Etelos Support page) generated by Marketplace module **6122** displayed to a user, so the user can edit an application or deploy an application. Field **17302** displays names of available applications; field **17304** displays the application type associated with an application; field **17306** displays the grouping information associated with an application; field **17308** displays an application version managed by the packager module **6136**. A user may archive an application by checking the checkboxes in field **17310**.

[0707] FIG. **174** is an exemplary screenshot **17400** of a support page (e.g. an Etelos Support page) generated by Marketplace module **6122**, wherein a user can learn about an application by reviewing the information in a knowledge base (e.g. an Etelos knowledge base). This knowledge base information is managed by marketplace database management module **6180**.

[0708] FIG. **175** is an exemplary screenshot **17500** of a support page (e.g. an Etelos Support page) generated by Marketplace module **6122** wherein installation of software application information associated with a user's account is displayed. Field **17502** displays install dates; field **17504** displays product information; field **17506** displays licensing information; field **17508** provides links to more installation information; field **17510** includes buttons for a user to cancel an installation or reinstall an installation.

[0709] FIG. **176** is an exemplary screenshot **17600** of a support page (e.g. an Etelos Support page) similar to FIG. **129** generated by marketplace module **6122**.

[0710] FIG. **177** is an exemplary screenshot **17700** of a support page (e.g. an Etelos Support page) generated by marketplace module **6122**. Discussion topics **17702** includes topics available in a forum page stored by marketplace database management module **6180**. Column **17704** includes discussion topics threads; column **17706** displays a date on which a forum thread was last posted; column **17708** displays number of replies to a forum thread; column **17710** displays number of times a forum thread has been viewed; column **17712** enables a user to subscribe to a forum thread by checking a checkbox.

[0711] FIG. **178** is an exemplary screenshot **17800** of a support page (e.g. an Etelos Support page) generated by Marketplace module **6122** that includes information on getting started for business users. This page is displayed to a user seeking support information regarding an application.

[0712] FIG. **179** is an exemplary screenshot **17900** of a support page (e.g. an Etelos Support page) generated by Marketplace module **6122**. Your installed applications **17902** enables a user to view the user's installed applications. Field **17904** includes links to other support areas e.g. getting started, account, forums, or knowledge base.

[0713] FIG. **180** is an exemplary screenshot **18000** of a support page (e.g. an Etelos Support page) generated by Marketplace module **6122**. Field **18002** includes information on an exemplary web developing application called EASE made available through Marketplace module **6122**. This page is accessed by an application developer/vendor using the marketplace (e.g. Etelos system).

[0714] FIG. **181** is an exemplary screenshot **18100** of a support page (e.g. an Etelos Support page) generated by Marketplace module **6122** wherein a user can modify the user's profile information. The information entered here is stored by user account access module **6172**.

[0715] FIG. **182** is an exemplary screenshot **18200** of a Marketplace homepage, generated by marketplace module **6122**.

[0716] FIG. **183** is an exemplary screenshot **18300** of a Marketplace page, generated by marketplace module **6122**, where an application is displayed for sale (e.g. Etelos Hosting and Dev Environment for BT Web21C SDK). This page shows reviews by users and a description of software applications.

[0717] FIG. **184** is an exemplary screenshot **18400** of a Marketplace page, generated by marketplace module **6122**, where an application is displayed for sale (e.g. Etelos Hosting and Dev Environment for BT Web21C SDK). In field Reviews **18402**, a user may post a review associated with an application. In field **18404** the user may type a title for the user's review; in field **18406** the user may type the text of the user's review. Radio buttons **18408-18416** enable the user to rate the user's satisfaction with different aspects of Marketplace services.

[0718] FIG. **185** is an exemplary screenshot **18500** of a Marketplace page, generated by marketplace module **6122**, where an application is displayed for sale (e.g. Etelos Hosting and Dev Environment for BT Web21C SDK). A user may purchase/license an application by clicking the button **18502** labeled Buy Now. A user may purchase/license a free trial of an application by clicking the button **18504** labeled Free Trial. The features tab **18506** is activated wherein information about features of the application are displayed.

[0719] FIG. **186** is an exemplary screenshot **18600** of a Marketplace page, generated by marketplace module **6122**, where an application is displayed for sale (e.g. Etelos Hosting and Dev Environment for BT Web21C SDK). A demos tab **18602** shows videos or screenshots of the application. This screen is displayed to a user looking for information about a software application.

[0720] FIG. **187** is an exemplary screenshot **18700** of a Marketplace page, generated by marketplace module **6122**, where an application is displayed for sale (e.g. Etelos Hosting and Dev Environment for BT Web21C SDK) and the FAQs tab **18702** is activated wherein user's frequently asked questions associated with the application are displayed.

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 229

[0721] FIG. **188** is an exemplary screenshot **18800** of a Marketplace page, generated by marketplace module **6122**, where an application is displayed for sale (e.g. Etelos Hosting and Dev Environment for BT Web21C SDK) and the forum tab **18802** is activated wherein forum discussions associated with the application are displayed.

[0722] FIG. **189** is an exemplary screenshot **18900** of a Marketplace page, generated by marketplace module **6122**, where an application is displayed for sale (e.g. Etelos Hosting and Dev Environment for BT Web21C SDK) and the support tab **18902** is activated wherein the support information associated with the application is displayed.

[0723] The foregoing description, for purpose of explanation, has been described with reference to specific embodiments. However, the illustrative discussions above are not intended to be exhaustive or to limit the invention to the precise forms disclosed. Many modifications and variations are possible in view of the above teachings. The embodiments were chosen and described in order to best explain the principles of the invention and its practical applications, to thereby enable others skilled in the art to best utilize the invention and various embodiments with various modifications as are suited to the particular use contemplated.

[0724] Start of **5006**: Software Marketplace and Distribution System Support for Foreign Prosecution.

[0725] 1. A computer implemented method, comprising:

[0726] at one or more servers, hosting a marketplace application:

[0727] receiving from a vendor a software application for distribution;

[0728] associating license terms with the software application;

[0729] making the software application available for distribution through the marketplace application; and

[0730] deploying the software application to one or more user accounts on one or more hosting servers, in accordance with the license terms.

[0731] 2. The computer-implemented method of claim **1**, further comprising presenting a description of the license terms to a user.

[0732] 3. The computer-implemented method of claim **1**, wherein the on one or more hosting servers are physically separate from the one or more servers hosting the marketplace application.

[0733] 4. The computer-implemented method of claim **1**, wherein the deploying is performed after a user request to deploy the software application.

[0734] 5. The computer-implemented method of claim **1**, wherein the deploying includes downloading the software to the one or more user accounts.

[0735] 6. The computer-implemented method of claim **1**, wherein the deploying includes activating a flag associated with the software application in the one or more user accounts.

[0736] 7. The computer-implemented method of claim **6**, where the flag enables for user the software application.

[0737] 8. The computer-implemented method of claim **1**, wherein the deploying includes activating a license for the software application in the one or more user accounts.

[0738] 9. The computer-implemented method of claim **1**, wherein the deploying includes providing the software application for hosting by a user on hosting servers associated with the user.

[0739] 10. The computer-implemented method of claim **1**, wherein distribution to a user through the marketplace application includes through a website associated with the marketplace application.

[0740] 11. The computer-implemented method of claim **1**, wherein distribution to a user through the marketplace application includes through a client associated with the marketplace application.

[0741] 12. The computer-implemented method of claim **1**, comprising packaging the software application for distribution via the marketplace application hosted by the one or more servers.

[0742] 13. The computer-implemented method of claim **12**, comprising storing a packaged software application in an application repository.

[0743] 14. The computer implemented method of claim **12** wherein packaging includes preparing an update to a previously deployed software application, where the update requires the previously deployed software application to function.

[0744] 15. The computer implemented method of claim **12** wherein packaging includes preparing a standalone distribution for a software application.

[0745] 16. The computer implemented method of claim **15** wherein the standalone distribution includes a software application and one or more updates to the application.

[0746] 17. The computer implemented method of claims **14**, **15**, **16** wherein the update is deployed to the one or more user accounts selected from the group consisting of a push method, a subscription (pull) method, and a hybrid method, in accordance with the license terms.

[0747] 18. The computer-implemented method of claim **1**, wherein the making available is performed by a listing manager.

[0748] 19. The computer implemented method of claim **18** wherein the listing manager includes a store listing for licensing the software application.

[0749] 20. The computer-implemented method of claim **1**, wherein the deploying is performed by an application deployer.

[0750] 21. The computer-implemented method of claim **1**, comprising hosting the deployed software application for the one or more user accounts.

[0751] 22. The computer implemented method of claims **1** or **19** wherein the marketplace application includes technical support for the software application.

[0752] 23. The computer implemented method of claims **1**, **19** or **22**, wherein making the software application available for distribution includes at least one selected from the group consisting of: determining a user account type, and based on the user account type, preparing to deploy a software application to the user account, or generating a new user account compatible with the software application and preparing to deploying the software application to the new user account.

[0753] 24. A server system, comprising:

[0754] one or more processors;

[0755] memory; and

[0756] one or more programs stored in the memory, the one or more programs comprising instructions for at one or more servers, host a marketplace application:

[0757] receiving from a vendor a software application for distribution;

[0758] associate license terms with the software application;

[0759] make the software application available for distribution through the marketplace application; and

[0760] deploy the software application to one or more user accounts on one or more hosting servers, in accordance with the license terms.

[0761] 25. The server system of claim **24**, further comprising instructions to present a description of the license terms to a user.

[0762] 26. The server system of claim **24**, wherein the on one or more hosting servers are physically separate from the one or more servers hosting the marketplace application.

[0763] 27. The server system of claim **24**, wherein the instructions to deploy include instructions to deploy the software application after a user request.

[0764] 28. The server system of claim **24**, wherein the instructions to deploy include instructions to download the software to the one or more user accounts.

[0765] 29. The server system of claim **24**, wherein the instructions to deploy include instructions to activate a flag associated with the software application in the one or more user accounts.

[0766] 30. The server system of claim **29**, where the flag enables for user the software application.

[0767] 31. The server system of claim **24**, wherein the instructions to deploy include instructions to activate a license for the software application in the one or more user accounts.

[0768] 32. The server system of claim **24**, wherein the instructions to deploy include instructions to provide the software application for hosting by a user on hosting servers associated with the user.

[0769] 33. The server system of claim **24**, wherein the instructions to distribute to a user through the marketplace application includes instructions to distribute through a website associated with the marketplace application.

[0770] 34. The server system of claim **24**, wherein the instructions to distribute to a user through the marketplace application includes instructions to distribute through a client associated with the marketplace application.

[0771] 35. The server system of claim **24**, comprising instructions to package the software application for distribution via the marketplace application hosted by the one or more servers.

[0772] 36. The server system of claim **35**, comprising instructions to store a packaged software application in an application repository.

[0773] 37. The server system of claim **35** wherein instructions to package include instructions to prepare an update to a previously deployed software application, where the update requires the previously deployed software application to function.

[0774] 38. The server system of claim **35** wherein instructions to package include instructions to preparing a standalone distribution for a software application.

[0775] 39. The server system of claim **38** wherein the standalone distribution includes a software application and one or more updates to the application.

[0776] 40. The server system of claims **37**, **38** or **39** where the update is deployed to the one or more user accounts selected from the group consisting of a push method, a subscription (pull) method, and a hybrid method, in accordance with the license terms.

[0777] 41. The server system of claim **24**, wherein the instructions to make available includes instructions for a listing manager.

[0778] 42. The server system of claim **41** wherein the listing manager includes a store listing for licensing the software application.

[0779] 43. The server system of claim **24**, wherein the instructions to deploy include instructions for an application deployer.

[0780] 44. The server system of claim **24**, comprising instructions to host the deployed software application for the one or more user accounts.

[0781] 45. The server system of claim **24** or **42** wherein the marketplace application includes technical support for the software application.

[0782] 46. The server system of claim **24**, **42**, or **45**, wherein instructions to make the software application available for distribution includes instructions for at least one selected from the group consisting of: determining a user account type, and based on the user account type, preparing to deploy a software application to the user account, or generating a new user account compatible with the software application and preparing to deploying the software application to the new user account.

[0783] 47. A computer readable storage medium storing one or more programs configured for execution by a computer, the one or more programs comprising instructions to at one or more servers, hosting a marketplace application:

[0784] receive from a vendor a software application for distribution;

[0785] associate license terms with the software application;

[0786] make the software application available for distribution through the marketplace application; and

[0787] deploy the software application to one or more user accounts on one or more hosting servers, in accordance with the license terms.

[0788] 48. A computer implemented method, comprising:

[0789] at one or more marketplace servers hosting a marketplace application, in response to a request from a syndicated server to distribute a software application from the marketplace:

[0790] identifying one or more user accounts associated with the request;

[0791] verifying that the one or more user accounts has permission to use the software application; and

[0792] deploying the software application to the one or more user accounts, in accordance with license terms associated with the software application.

[0793] 49. The computer-implemented method of claim **48**, wherein deploying includes presenting for deployment to a user.

[0794] 50. The computer-implemented method of claim **48**, comprising making the software application available for distribution through the syndicated server.

[0795] 51. The computer-implemented method of claim **48** or **49**, wherein deploying includes providing through an application deployer, across a network to the one or more user accounts, a software application stored at the application repository.

[0796] 52. The computer-implemented method of claim **51**, wherein deploying includes selecting one of a plurality of software applications, compatible with the one or more user accounts, from the application repository.

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 231

[0797] 53. The computer-implemented method of claim **51** or **52**, wherein the application repository stores software applications in a plurality of states, including at least one selected from the group consisting of a ready to deploy state, an undergoing quality assurance state, a ready to submit for quality assurance state, and an unfinished state.

[0798] 54. A server system, comprising:

[0799] one or more processors;

[0800] memory; and

[0801] one or more programs stored in the memory, the one or more programs comprising instructions to at one or more marketplace servers hosting a marketplace application, in response to a request from a syndicated server to distribute a software application from the marketplace:

[0802] identify one or more user accounts associated with the request;

[0803] verify that the one or more user accounts has permission to use the software application; and

[0804] deploy the software application to the one or more user accounts, in accordance with license terms associated with the software application.

[0805] 55. The server system of claim **54**, wherein instructions to deploy include instructions to present for deployment to a user.

[0806] 56. The server system of claim **54**, further comprising instructions to make the software application available for distribution through the syndicated server.

[0807] 57. The server system of claim **54** or **55**, wherein instructions to deploy include instructions to provide through an application deployer, across a network to the one or more user accounts, a software application stored at the application repository.

[0808] 58. The server system of claim **57**, wherein instructions to deploy include instructions to select one of a plurality of software applications, compatible with the one or more user accounts, from the application repository.

[0809] 59. The server system of claim **57** or **58**, wherein the application repository stores software applications in a plurality of states, including at least one selected from the group consisting of a ready to deploy state, an undergoing quality assurance state, a ready to submit for quality assurance state, and an unfinished state.

[0810] 60. A computer readable storage medium storing one or more programs configured for execution by a computer, the one or more programs comprising instructions for at one or more marketplace servers hosting a marketplace application, in response to a request from a syndicated server to distribute a software application from the marketplace:

[0811] identifying one or more user accounts associated with the request;

[0812] verifying that the one or more user accounts has permission to use the software application; and

[0813] deploying the software application to the one or more user accounts, in accordance with license terms associated with the software application.

[0814] Data Structure for purchasing, deploying, hosting programs:

[0815] 61. A computer system, comprising:

[0816] one or more processors;

[0817] memory; and

[0818] one or more programs stored in the memory, the one or more programs comprising instructions for implementing:

[0819] a program module configured to provide a software application for distribution in response to an access request from a user;

[0820] a program module configured to receive and deploy the software application from the marketplace module to an account on one or more servers; and

[0821] a program module configured to provide at least one or more user accounts, from which the user accesses the software application.

[0822] 62. The system of claim **61**, wherein the one or more user accounts receives a link to the deployed software application, wherein the link permits the one or more user accounts to access the deployed software application.

[0823] 63. The system of claim **61**, wherein the program module configured to provide a software application for distribution is a marketplace module.

[0824] 64. The system of claim **61**, wherein the program module configured to provide at least one or more user accounts is a one or more user accounts module

[0825] 65. The system of claim **61**, wherein the program module configured to receive and deploy a software application to the one or more user accounts module is a deployment module.

[0826] 66. The system of claim **61**, comprising an application repository storing software applications ready for distribution.

[0827] 67. The system of claim **61**, comprising a licensing module configured to provide a license associated with the software application, and to ensure that the deploying is performed in accordance with the license.

[0828] 68. The system of claim **64**, wherein the licensing module is configured to verify user permission to access a deployed software application prior to executing the application.

[0829] 69. The system of claim **61**, comprising a billing module configured to receive payment associated with the one or more user accounts for deployment of the selected software application.

[0830] 70. The system of claim **69**, wherein the billing module is configured to receive subscription payments associated with the one or more user accounts.

[0831] 71. The system of claim **69**, wherein an amount of the payment varies in accordance with license terms associated with the software application.

[0832] 72. The system of claim **69**, wherein the billing module is configured to receive from the user a promotional code prior to processing a payment, and the billing module is configured to process the payment based on the promotional code.

[0833] 73. The system of claim **61**, comprising a listing manager module for managing marketplace content related to the software application.

[0834] 74. The system of claim **61**, comprising a deployer module for accessing the software application from the application repository and deploying the software application in the hosting infrastructure module.

[0835] 75. The system of claim **61**, wherein the access request comprises a selection instruction for a software application compatible with a framework associated with the one or more user accounts.

[0836] 76. The system of claim **61**, wherein the user accesses the software application through a one or more user accounts registered with the hosting infrastructure module.

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 232

[0837] 77. The system of claim **61**, wherein following installation, the hosting infrastructure module sends an installation confirmation to the marketplace module.

[0838] 78. The computer implemented method of any one of claims **61** to **77**, wherein the software application is a web-based software application, executed at one or more servers.

[0839] End of **5006**: Software Marketplace and Distribution System Support for Foreign Prosecution.

[0840] Start of **5007**: Software Licensing and Enforcement System support for Foreign Prosecution.

[0841] 1. A computer implemented method, comprising:

[0842] at one or more servers, hosting a marketplace application:

[0843] receiving from a vendor a software application for distribution;

[0844] generating license terms in response to a selection by the vendor from options provided by the marketplace application;

[0845] associating the license terms with the software application, and

[0846] making the software application available for distribution through the marketplace application, in accordance with the license terms.

[0847] 2. The computer implemented method of claim **1**, further comprising deploying the software application to one or more user accounts on one or more hosting servers, in accordance with the license terms.

[0848] 3. The computer implemented method of claim **2**, where the deploying is in response to a payment associated with the one or more user accounts.

[0849] 4. The computer implemented method of claim **1**, wherein the license terms include at least one of an open source license, a closed license, a source code license, an executable license, and a repacking license.

[0850] 5. The computer implemented method of claim **4**, wherein the repacking license determines whether a user of the software application is permitted to repackage and redistribute the software application.

[0851] 6. The computer implemented method of claim **5**, wherein the repacking license has an associated royalty.

[0852] 7. The computer implemented method of claim **6**, wherein the associated royalty is one selected from the group consisting of a wholesale royalty, a retail royalty, and a flat fee.

[0853] 8. The computer implemented method of claims **1**, **4**, or **5**, wherein a license manager determines user permissions for installation, activation, and access to features of applications.

[0854] 9. The computer implemented method of claim **1** wherein the options provided by the marketplace application includes an option to use license terms supplied by the vendor.

[0855] 10. The computer implemented method of claims **1**, **4**, **5**, **8**, or **9** including displaying licensing events for a respective software application made available for distribution through the marketplace application.

[0856] 11. The computer-implemented method of claim **10**, wherein displaying includes displaying the licensing events to a respective vendor associated with the respective software application.

[0857] 12. The computer-implemented method of claim **1**, comprising storing the license terms in a licensing manager.

[0858] 13. The computer implemented method of claim **1**, wherein a price associated with the software application is stored at a licensing manager separately from the software application.

[0859] 14. The computer implemented method of claim **13**, wherein the price is dynamically adjusted by the licensing manager in response to a selection by the software vendor.

[0860] 15. The computer implemented method of claim **1**, wherein at least one selected from the group consisting of access duration, features, and price, is dynamically adjusted by a licensing manager in response to a selection by the software vendor.

[0861] 16. The computer implemented method of claim **1**, **2**, wherein the one or more user accounts are stored separately from the marketplace application.

[0862] 17. The computer implemented method of claims **1** or **16** further comprising processing a payment associated with the software application.

[0863] 18. The computer implemented method of claim **17**, further comprising, prior to processing a payment, receiving from the user a promotional code, and processing the payment based on the promotional code.

[0864] 19. The computer implemented method of claim **17**, comprising storing a record of the processed payment in a billing record.

[0865] 20. The computer implemented method of claim **19**, comprising prior to executing the deployed application, comparing a user identifier associated with the one or more user accounts and an application id associated with the deployed application against a billing manager to verify that a valid payment has been recorded.

[0866] 21. The computer implemented methods of claims **1**, **16**, **17**, or **20**, comprising verifying that the one or more user accounts has permission to execute the deployed application, and in the event of a verification failure, warning the user.

[0867] 22. The computer implemented method of claim **21**, wherein the verifying is performed periodically, and following a plurality of verification failures, preventing the one or more user accounts from executing the deployed application.

[0868] 23. The computer implemented method of claims **21** or **22**, wherein the verifying includes checking for multiple instances of the software application being simultaneously executed by the one or more user accounts.

[0869] 24. The computer implemented method of claim **16**, comprising preventing access by the user to data stored at the one or more servers, upon determining that the one or more user accounts has been disabled.

[0870] 25. The computer implemented method of claim **16**, further comprising deploys a license key associated with the software application to a user account associated with a licensee of the software application.

[0871] 26. The computer implemented method of claim **25**, further comprising communicating to the software vendor that the software application has been licensed by the user.

[0872] 27. The computer implemented method of claim **26**, wherein the communicating is performed via an application programming interface call.

[0873] 28. A server system, comprising:

[0874] one or more processors;

[0875] memory; and

[0876] one or more programs stored in the memory, the one or more programs comprising instructions for, at one or more servers hosting a marketplace application:

[0877] receiving from a vendor a software application for distribution;

[0878] generating license terms in response to a selection by the vendor from options provided by the marketplace application;

[0879] associating the license terms with the software application, and

[0880] making the software application available for distribution through the marketplace application, in accordance with the license terms.

[0881] 29. The computer implemented method of claim **28**, further comprising deploying the software application to one or more user accounts on one or more hosting servers, in accordance with the license terms.

[0882] 30. The computer implemented method of claim **29**, where the deploying is in response to a payment associated with the one or more user accounts.

[0883] 31. The computer implemented method of claim **28**, wherein the license terms include at least one of an open source license, a closed license, a source code license, an executable license, and a repacking license.

[0884] 32. The computer implemented method of claim **28**, wherein the repacking license determines whether a user of the software application is permitted to repackage and redistribute the software application.

[0885] 33. The computer implemented method of claim **32**, wherein the repacking license has an associated royalty.

[0886] 34. The computer implemented method of claim **33**, wherein the associated royalty is one selected from the group consisting of a wholesale royalty, a retail royalty, and a flat fee.

[0887] 35. The computer implemented method of claims **28**, **32**, or **33**, wherein a license manager determines user permissions for installation, activation, and access to features of applications.

[0888] 36. The computer implemented method of claim **28** wherein the options provided by the marketplace application includes an option to use license terms supplied by the vendor.

[0889] 37. The computer implemented method of claims **28**, **32**, **33**, **35**, or **36** including displaying licensing events for a respective software application made available for distribution through the marketplace application.

[0890] 38. The computer-implemented method of claim **37**, wherein displaying includes displaying the licensing events to a respective vendor associated with the respective software application.

[0891] 39. The computer-implemented method of claim **28**, comprising storing the license terms in a licensing manager.

[0892] 40. The computer implemented method of claim **28**, wherein a price associated with the software application is stored at a licensing manager separately from the software application.

[0893] 41. The computer implemented method of claim **40**, wherein the price is dynamically adjusted by the licensing manager in response to a selection by the software vendor.

[0894] 42. The computer implemented method of claim **28**, wherein at least one selected from the group consisting of access duration, features, and price, is dynamically adjusted by a licensing manager in response to a selection by the software vendor.

[0895] 43. The computer implemented method of claim **29**, wherein the one or more user accounts are stored separately from the marketplace application.

[0896] 44. The computer implemented method of claims **28** or **43** further comprising processing a payment associated with the software application.

[0897] 45. The computer implemented method of claim **44**, further comprising, prior to processing a payment, receiving from the user a promotional code, and processing the payment based on the promotional code.

[0898] 46. The computer implemented method of claim **44**, comprising storing a record of the processed payment in a billing record.

[0899] 47. The computer implemented method of claim **44**, comprising prior to executing the deployed application, comparing a user identifier associated with the one or more user accounts and an application id associated with the deployed application against a billing manager to verify that a valid payment has been recorded.

[0900] 48. The computer implemented methods of claims **28**, **43**, **44**, and **47** comprising verifying that the one or more user accounts has permission to execute the deployed application, and in the event of a verification failure, warning the user.

[0901] 49. The computer implemented method of claim **48**, wherein the verifying is performed periodically, and following a plurality of verification failures, preventing the one or more user accounts from executing the deployed application.

[0902] 50. The computer implemented method of claims **48** or **49**, wherein the verifying includes checking for multiple instances of the software application being simultaneously executed by the one or more user accounts.

[0903] 51. The computer implemented method of claim **43**, comprising preventing access by the user to data stored at the one or more servers, upon determining that the one or more user accounts has been disabled.

[0904] 52. The computer implemented method of claim **43**, further comprising deploys a license key associated with the software application to a user account associated with a licensee of the software application.

[0905] 53. The computer implemented method of claim **52**, further comprising communicating to the software vendor that the software application has been licensed by the user.

[0906] 54. The computer implemented method of claim **53**, wherein the communicating is performed via an application programming interface call.

[0907] 55. A computer readable storage medium storing one or more programs configured for execution by a computer, the one or more programs comprising instructions for

[0908] at one or more servers, hosting a marketplace application:

[0909] receiving from a vendor a software application for distribution;

[0910] generating license terms in response to a selection by the vendor from options provided by the marketplace application;

[0911] associating the license terms with the software application, and

[0912] making the software application available for distribution through the marketplace application, in accordance with the license terms.

[0913] 56. A computer implemented method, comprising:

[0914] at one or more servers hosting a marketplace application:

[0915] making a software application available for distribution through the marketplace application;

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 234

[0916] receiving a user request to license the software application; and

[0917] providing the software application for deployment to one or more user accounts, hosted on one or more servers.

[0918] 57. The computer implemented method of claim **56**, where providing the software application for deployment includes providing the software application across a network for deployment at one or more servers associated with the user, hosting the one or more user accounts.

[0919] 58. The computer implemented method of claim **56**, further comprising processing a selection by the user that includes adding the software application to a cart associated with the user, and checking out the cart.

[0920] 59. The computer implemented method of claim **56**, comprising prior to deploying, presenting a description of license terms associated with the software application to the user.

[0921] 60. The computer implemented method of claim **57**, wherein the license terms are specified by a vendor associated with the software application.

[0922] 61. The computer implemented method of claim **60**, wherein specifying the license terms includes selecting the license terms from a plurality of options provided by a licensing engine associated with the one or more servers hosting a marketplace application.

[0923] 62. The computer implemented method of claim **57**, **60**, or **61** comprising validating that the request to license the software application complies with the license terms.

[0924] 63. The computer implemented method of claim **56**, **57**, **60**, **61** or **62** wherein the request to license the software application includes a payment selected from the group consisting of a cash payment, a credit payment, and a prospective future payment.

[0925] 64. The computer implemented method of claim **56**, **57**, **60**, **61**, **62** or **63** wherein the software application is a web-based software application, executed at one or more servers.

[0926] 65. A server system, comprising:

[0927] one or more processors;

[0928] memory; and

[0929] one or more programs stored in the memory, the one or more programs comprising instructions to at one or more servers hosting a marketplace application:

[0930] make a software application available for distribution through the marketplace application;

[0931] receive a user request to license the software application; and

[0932] provide the software application for deployment to one or more user accounts, hosted on one or more servers.

[0933] 66. The server system of claim **65**, wherein the instructions to provide the software application for deployment includes instructions to provide the software application across a network for deployment at one or more servers associated with the user, hosting the one or more user accounts.

[0934] 67. The server system of claim **65**, further comprising instructions to process a selection by the user that includes adding the software application to a cart associated with the user, and checking out the cart.

[0935] 68. The server system of claim **65**, further comprising instructions for prior to deploying, presenting a description of license terms associated with the software application to the user.

[0936] 69. The server system of claim **68**, further comprising instructions to receive license terms specified by a vendor associated with the software application.

[0937] 70. The server system of claim **69**, wherein receiving license terms specified by a vendor include receiving a selection of license terms from a plurality of options provided by a licensing engine associated with the one or more servers hosting a marketplace application.

[0938] 71. The server system of claim **66**, **69**, or **70** further comprising instructions to validate that the request to license the software application complies with the license terms.

[0939] 72. The server system of claims **65**, **66**, **69**, **70**, or **71** wherein the request to license the software application includes a payment selected from the group consisting of a cash payment, a credit payment, and a prospective future payment.

[0940] 73. The server system of claims **65**, **66**, **69**, **70**, **71**, or **72** wherein the software application is a web-based software application, executed at one or more servers.

[0941] 74. A computer readable storage medium storing one or more programs configured for execution by a computer, the one or more programs comprising instructions for at one or more servers hosting a marketplace application:

[0942] making a software application available for distribution through the marketplace application;

[0943] receiving a user request to license the software application; and

[0944] providing the software application for deployment to one or more user accounts, hosted on one or more servers.

[0945] End of **5007**: Software Licensing and Enforcement System Support for Foreign Prosecution.

What is claimed is:

1. A computer implemented method, comprising:

at one or more servers, hosting a marketplace application:

receiving from a vendor a software application for distribution;

generating license terms in response to a selection by the vendor from options provided by the marketplace application;

associating the license terms with the software application, and

making the software application available for distribution through the marketplace application, in accordance with the license terms.

2. The computer implemented method of claim **1**, further comprising deploying the software application to one or more user accounts on one or more hosting servers, in accordance with the license terms.

3. The computer implemented method of claim **2**, where the deploying is in response to a payment associated with the one or more user accounts.

4. The computer implemented method of claim **1**, wherein the license terms include at least one of an open source license, a closed license, a source code license, an executable license, and a repacking license.

5. The computer implemented method of claim **4**, wherein the repacking license determines whether a user of the software application is permitted to repackage and redistribute the software application.

6. The computer implemented method of claim **5**, wherein the repacking license has an associated royalty.

7. The computer implemented method of claim **6**, wherein the associated royalty is one selected from the group consisting of a wholesale royalty, a retail royalty, and a flat fee.

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 235

8. The computer implemented method of claim **1**, wherein a license manager determines user permissions for installation, activation, and access to features of applications.

9. The computer implemented method of claim **1** wherein the options provided by the marketplace application includes an option to use license terms supplied by the vendor.

10. The computer implemented method of claim **1**, including displaying licensing events for a respective software application made available for distribution through the marketplace application.

11. The computer-implemented method of claim **10**, wherein displaying includes sending for display the licensing events to a respective vendor associated with the respective software application.

12. The computer-implemented method of claim **1**, further comprising storing the license terms in a licensing manager.

13. The computer implemented method of claim **1**, wherein a price associated with the software application is stored at a licensing manager separately from the software application.

14. The computer implemented method of claim **13**, wherein the price is dynamically adjusted by the licensing manager in response to a selection by the software vendor.

15. The computer implemented method of claim **1**, wherein at least one selected from the group consisting of access duration, features, and price, is dynamically adjusted by a licensing manager in response to a selection by the software vendor.

16. The computer implemented method of claim **1**, wherein the one or more user accounts are stored separately from the marketplace application.

17. The computer implemented method of claim **1** further comprising processing a payment associated with the software application.

18. The computer implemented method of claim **17**, further comprising, prior to processing a payment, receiving from the user a promotional code, and processing the payment based on the promotional code.

19. The computer implemented method of claim **17**, further comprising storing a record of the processed payment in a billing record.

20. The computer implemented method of claim **19**, further comprising prior to executing the deployed application, comparing a user identifier associated with the one or more user accounts and an application id associated with the deployed application against a billing manager to verify that a valid payment has been recorded.

21. The computer implemented methods of claim **1**, further comprising verifying that the one or more user accounts has permission to execute the deployed application, and in the event of a verification failure, warning the user.

22. The computer implemented method of claim **21**, wherein the verifying is performed periodically, and following a plurality of verification failures, preventing the one or more user accounts from executing the deployed application.

23. The computer implemented method of claims **21**, wherein the verifying includes checking for multiple instances of the software application being simultaneously executed by the one or more user accounts.

24. The computer implemented method of claim **16**, further comprising preventing access by the user to data stored at the one or more servers, upon determining that the one or more user accounts has been disabled.

25. The computer implemented method of claim **16**, further comprising deploying a license key associated with the software application to a user account associated with a licensee of the software application.

26. The computer implemented method of claim **25**, further comprising communicating to the software vendor that the software application has been licensed by the user.

27. The computer implemented method of claim **26**, wherein the communicating is performed via an application programming interface call.

28. A server system, comprising:

one or more processors;

memory; and

one or more programs stored in the memory, the one or more programs comprising instructions for, at one or more servers hosting a marketplace application:

receiving from a vendor a software application for distribution;

generating license terms in response to a selection by the vendor from options provided by the marketplace application;

associating the license terms with the software application, and

making the software application available for distribution through the marketplace application, in accordance with the license terms.

29. The server system of claim **28**, further comprising instructions for deploying the software application to one or more user accounts on one or more hosting servers, in accordance with the license terms.

30. The server system of claim **29**, where the deploying is in response to a payment associated with the one or more user accounts.

31. The server system of claim **28**, wherein the license terms include at least one of an open source license, a closed license, a source code license, an executable license, and a repacking license.

32. The server system of claim **28**, wherein the repacking license determines whether a user of the software application is permitted to repackage and redistribute the software application.

33. The server system of claim **32**, wherein the repacking license has an associated royalty.

34. The server system of claim **33**, wherein the associated royalty is one selected from the group consisting of a wholesale royalty, a retail royalty, and a flat fee.

35. The server system of claim **28**, further comprising instructions for license manager to determines user permissions for installation, activation, and access to features of applications.

36. The server system of claim **28** wherein the options provided by the marketplace application includes an option to use license terms supplied by the vendor.

37. The server system of claim **28**, including instructions to display licensing events for a respective software application made available for distribution through the marketplace application.

38. The server system of claim **37**, wherein displaying includes sending for display the licensing events to a respective vendor associated with the respective software application.

39. The server system of claim **28**, further comprising instructions to store the license terms in a licensing manager.

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 236

40. The server system of claim **28**, wherein a price associated with the software application is stored at a licensing manager separately from the software application.

41. The server system of claim **40**, wherein the price is dynamically adjusted by the licensing manager in response to a selection by the software vendor.

42. The server system of claim **28**, wherein at least one selected from the group consisting of access duration, features, and price, is dynamically adjusted by a licensing manager in response to a selection by the software vendor.

43. The server system of claim **29**, wherein the one or more user accounts are stored separately from the marketplace application.

44. The server system of claim **28** further comprising instructions to process a payment associated with the software application.

45. The server system of claim **44**, further comprising instructions for, prior to processing a payment, receiving from the user a promotional code, and processing the payment based on the promotional code.

46. The server system of claim **44**, further comprising instructions for storing a record of the processed payment in a billing record.

47. The server system of claim **44**, further comprising instructions for prior to executing the deployed application, comparing a user identifier associated with the one or more user accounts and an application id associated with the deployed application against a billing manager to verify that a valid payment has been recorded.

48. The server system of claim **28**, further comprising instructions for verifying that the one or more user accounts has permission to execute the deployed application, and in the event of a verification failure, warning the user.

49. The server system of claim **48**, wherein the verifying is performed periodically, and following a plurality of verification failures, preventing the one or more user accounts from executing the deployed application.

50. The server system of claim **48**, wherein the verifying includes checking for multiple instances of the software application being simultaneously executed by the one or more user accounts.

51. The server system of claim **43**, further comprising instructions for preventing access by the user to data stored at the one or more servers, upon determining that the one or more user accounts has been disabled.

52. The server system of claim **43**, further comprising instructions for deploying a license key associated with the software application to a user account associated with a licensee of the software application.

53. The server system of claim **52**, further comprising instructions for communicating to the software vendor that the software application has been licensed by the user.

54. The server system of claim **53**, wherein the communicating is performed via an application programming interface call.

55. A computer readable storage medium storing one or more programs configured for execution by a computer, the one or more programs comprising instructions for

at one or more servers, hosting a marketplace application:

receiving from a vendor a software application for distribution;

generating license terms in response to a selection by the vendor from options provided by the marketplace application;

associating the license terms with the software application, and

making the software application available for distribution through the marketplace application, in accordance with the license terms.

56. A computer implemented method, comprising:

at one or more servers hosting a marketplace application:

making a software application available for distribution through the marketplace application;

receiving a user request to license the software application; and

providing the software application for deployment to one or more user accounts, hosted on one or more servers.

57. The computer implemented method of claim **56**, where providing the software application for deployment includes providing the software application across a network for deployment at one or more servers associated with the user, hosting the one or more user accounts.

58. The computer implemented method of claim **56**, further comprising processing a selection by the user that includes adding the software application to a cart associated with the user, and checking out the cart.

59. The computer implemented method of claim **56**, further comprising prior to deploying, presenting a description of license terms associated with the software application to the user.

60. The computer implemented method of claim **57**, wherein the license terms are specified by a vendor associated with the software application.

61. The computer implemented method of claim **60**, wherein specifying the license terms includes selecting the license terms from a plurality of options provided by a licensing engine associated with the one or more servers hosting a marketplace application.

62. The computer implemented method of claim **57**, further comprising validating that the request to license the software application complies with the license terms.

63. The computer implemented method of claim **56**, wherein the request to license the software application includes a payment selected from the group consisting of a cash payment, a credit payment, and a prospective future payment.

64. The computer implemented method of claim **56**, wherein the software application is a web-based software application, executed at one or more servers.

65. A server system, comprising:

one or more processors;

memory; and

one or more programs stored in the memory, the one or more programs comprising instructions to at one or more servers hosting a marketplace application:

make a software application available for distribution through the marketplace application;

receive a user request to license the software application; and

provide the software application for deployment to one or more user accounts, hosted on one or more servers.

66. The server system of claim **65**, wherein the instructions to provide the software application for deployment includes instructions to provide the software application across a network for deployment at one or more servers associated with the user, hosting the one or more user accounts.

67. The server system of claim **65**, further comprising instructions to process a selection by the user that includes

Apple v. Uniloc, IPR2017-2202
Uniloc's Exhibit 2002, page 237

adding the software application to a cart associated with the user, and checking out the cart.

68. The server system of claim **65**, further comprising instructions for prior to deploying, presenting a description of license terms associated with the software application to the user.

69. The server system of claim **68**, further comprising instructions to receive license terms specified by a vendor associated with the software application.

70. The server system of claim **69**, wherein instructions to receive license terms specified by a vendor include instructions to receive a selection of license terms from a plurality of options provided by a licensing engine associated with the one or more servers hosting a marketplace application.

71. The server system of claim **66**, further comprising instructions to validate that the request to license the software application complies with the license terms.

72. The server system of claim **65**, wherein the request to license the software application includes a payment selected from the group consisting of a cash payment, a credit payment, and a prospective future payment.

73. The server system of claim **65**, wherein the software application is a web-based software application, executed at one or more servers.

74. A computer readable storage medium storing one or more programs configured for execution by a computer, the one or more programs comprising instructions for at one or more servers hosting a marketplace application:

making a software application available for distribution through the marketplace application;

receiving a user request to license the software application; and

providing the software application for deployment to one or more user accounts, hosted on one or more servers.

* * * * *