US 20030056207A1

(19) United States

(12) Patent Application Publication (10) Pub. No.: US 2003/0056207 A1

Fischer et al. (43) Pub. Date: Mar. 20, 2003

(54) PROCESS FOR DEPLOYING SOFTWARE FROM A CENTRAL COMPUTER SYSTEM TO REMOTELY LOCATED DEVICES

(76) Inventors: **Claudius Fischer**, Schwetzingen (DE); **Thorsten Stephan**, Heidelberg (DE); **Markus Schmidt-Karaca**, Heidelberg (DE); **Jochen Hertweck**, Schwetzingen (DE); **Franz Muller**, Stutensee (DE)

Correspondence Address:
Timothy F. Loomis
Law Offices of Timothy F. Loomis
2932 Hagen Drive
Plano, TX 75025 (US)

(21) Appl. No.: **10/164,248**

(22) Filed: **Jun. 5, 2002**

Related U.S. Application Data

(60) Provisional application No. 60/296,234, filed on Jun. 6, 2001. Provisional application No. 60/296,993, filed on Jun. 8, 2001.

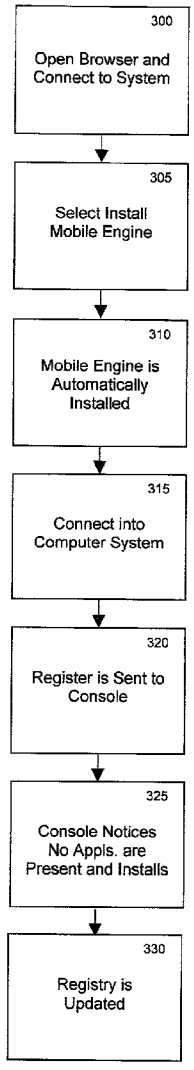
Publication Classification

(51) **Int. Cl.**[7] **G06F 9/445**

(52) **U.S. Cl.** **717/174**

(57) **ABSTRACT**

A process for deploying and installing software, including a platform-independent framework and offline applications, on devices, such as a personal digital assistant, Internet-enabled phone, handheld computer, laptop computer or desktop computer, remotely located from a central computer system. The framework and applications permitting the devices to run applications offline and synchronize data with the central computer system.



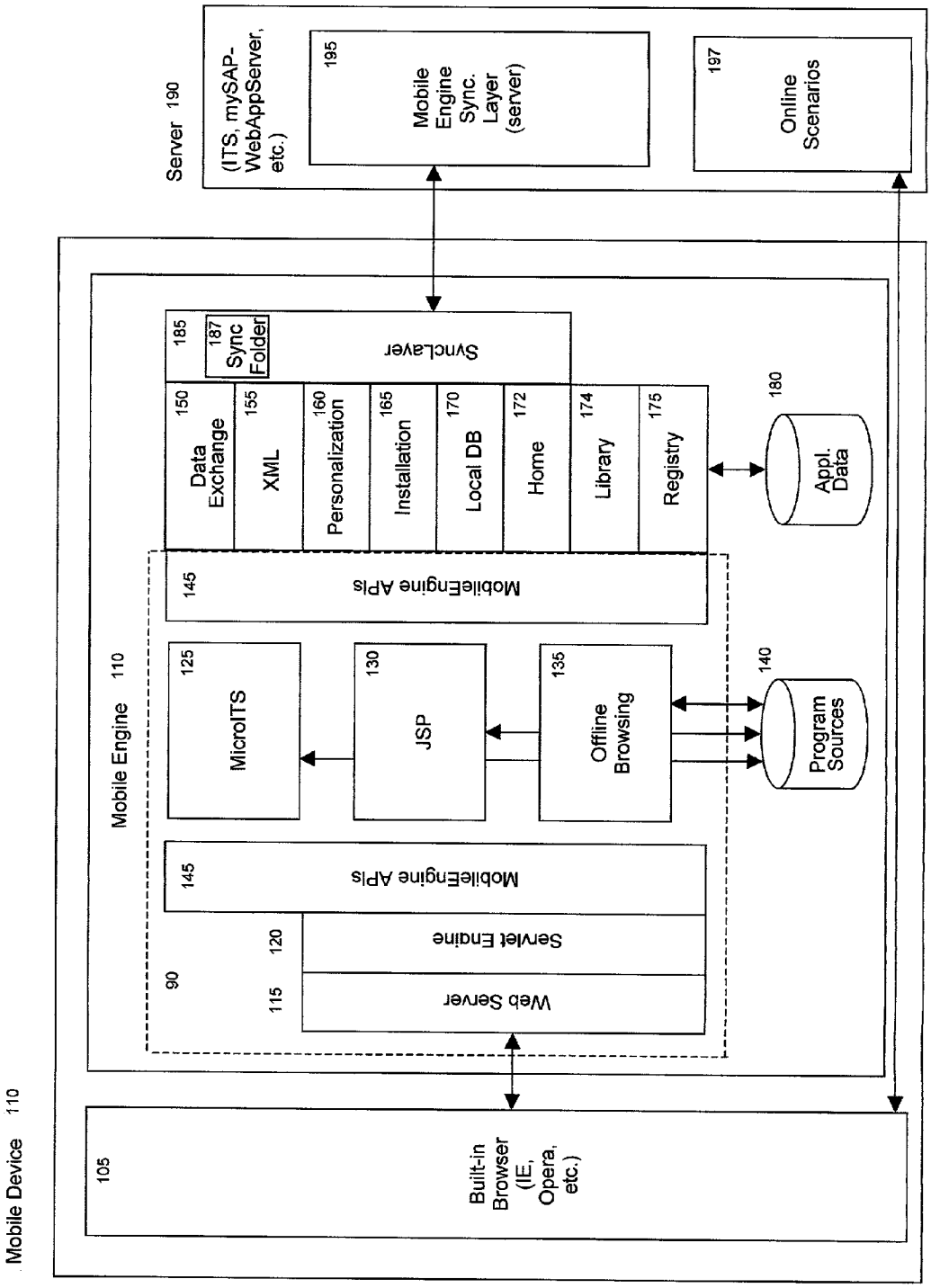

Server 190
(ITS, mySAP-WebAppServer, etc.)
195
Mobile Engine Sync. Layer (server)
197
Online Scenarios
Mobile Device 110
105
Built-in Browser (IE, Opera, etc.)
Mobile Engine 110
185
SyncLayer
187
Sync Folder
150
Data Exchange
155
XML
160
Personalization
165
Installation
170
Local DB
172
Home
174
Library
175
Registry
180
Appl. Data
145
MobileEngine APIs
125
MicroITS
130
JSP
135
Offline Browsing
140
Program Sources
145
MobileEngine APIs
90
120
Servlet Engine
115
Web Server

Fig. 1

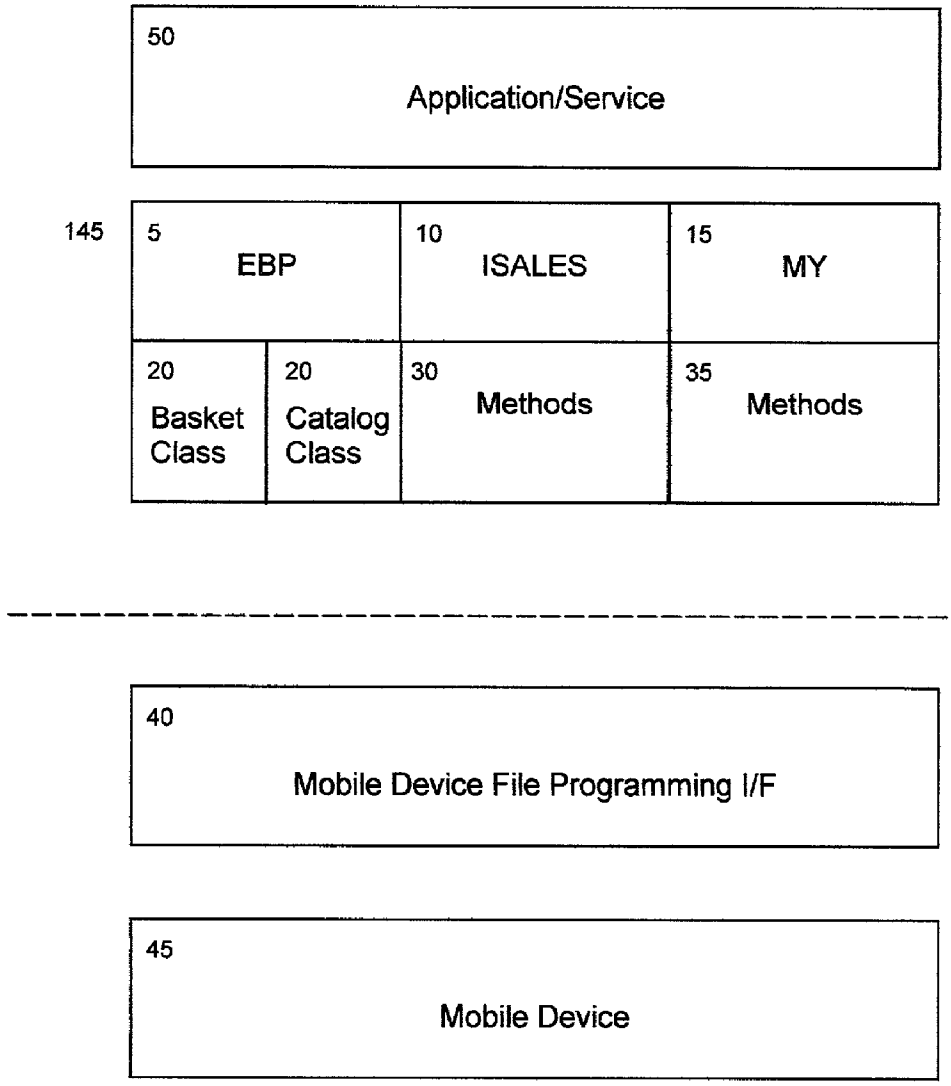

50
Application/Service
145
5
EBP
10
ISALES
15
MY
20
Basket Class
20
Catalog Class
30
Methods
35
Methods
40
Mobile Device File Programming I/F
45
Mobile Device

Fig. 2

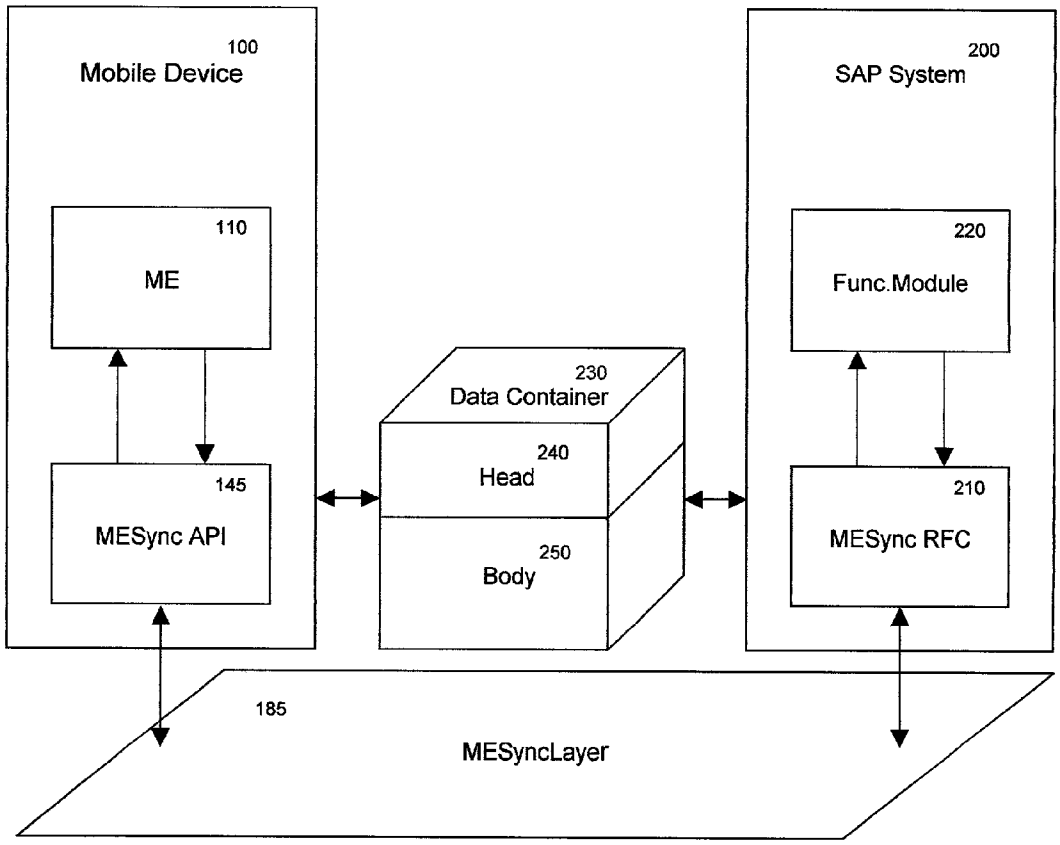

100
Mobile Device
110
ME
145
MESync API
230
Data Container
240
Head
250
Body
200
SAP System
220
Func.Module
210
MESync RFC
185
MESyncLayer

Fig. 3

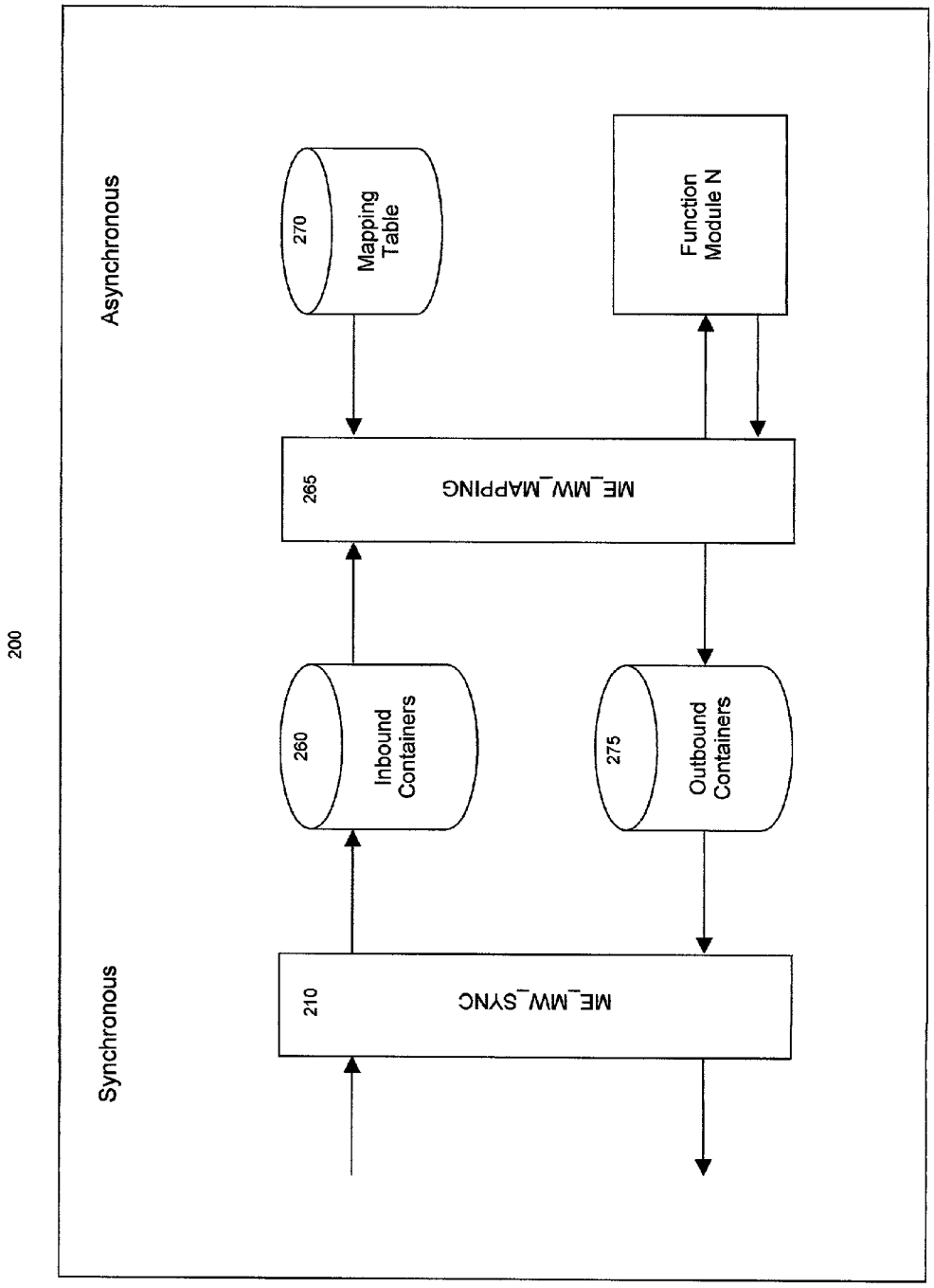

200
Asynchronous
Synchronous
270
Mapping Table
Function Module N
265
ME_MW_MAPPING
260
Inbound Containers
275
Outbound Containers
210
ME_MW_SYNC

Fig. 4

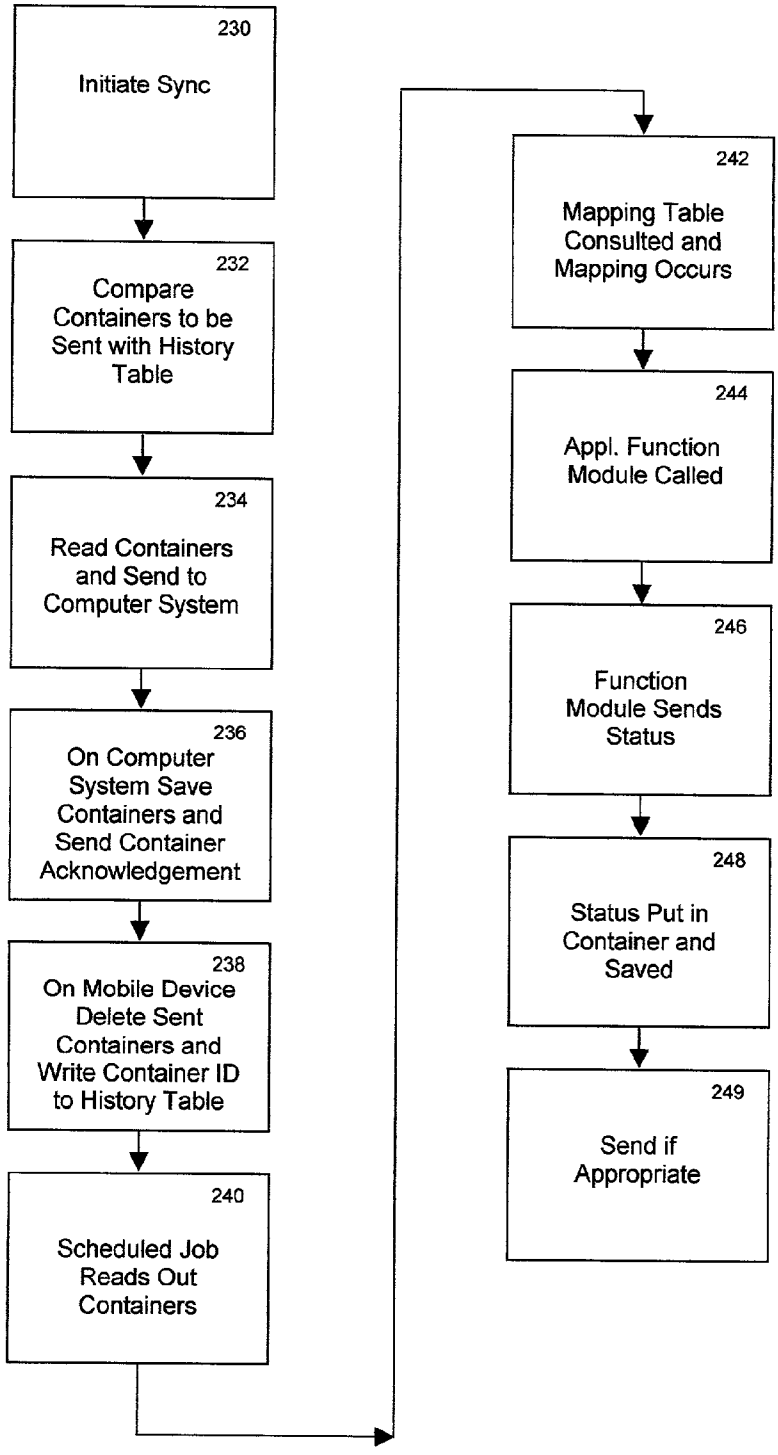

230
Initiate Sync
232
Compare Containers to be Sent with History Table
234
Read Containers and Send to Computer System
236
On Computer System Save Containers and Send Container Acknowledgement
238
On Mobile Device Delete Sent Containers and Write Container ID to History Table
240
Scheduled Job Reads Out Containers
242
Mapping Table Consulted and Mapping Occurs
244
Appl. Function Module Called
246
Function Module Sends Status
248
Status Put in Container and Saved
249
Send if Appropriate

Fig. 5

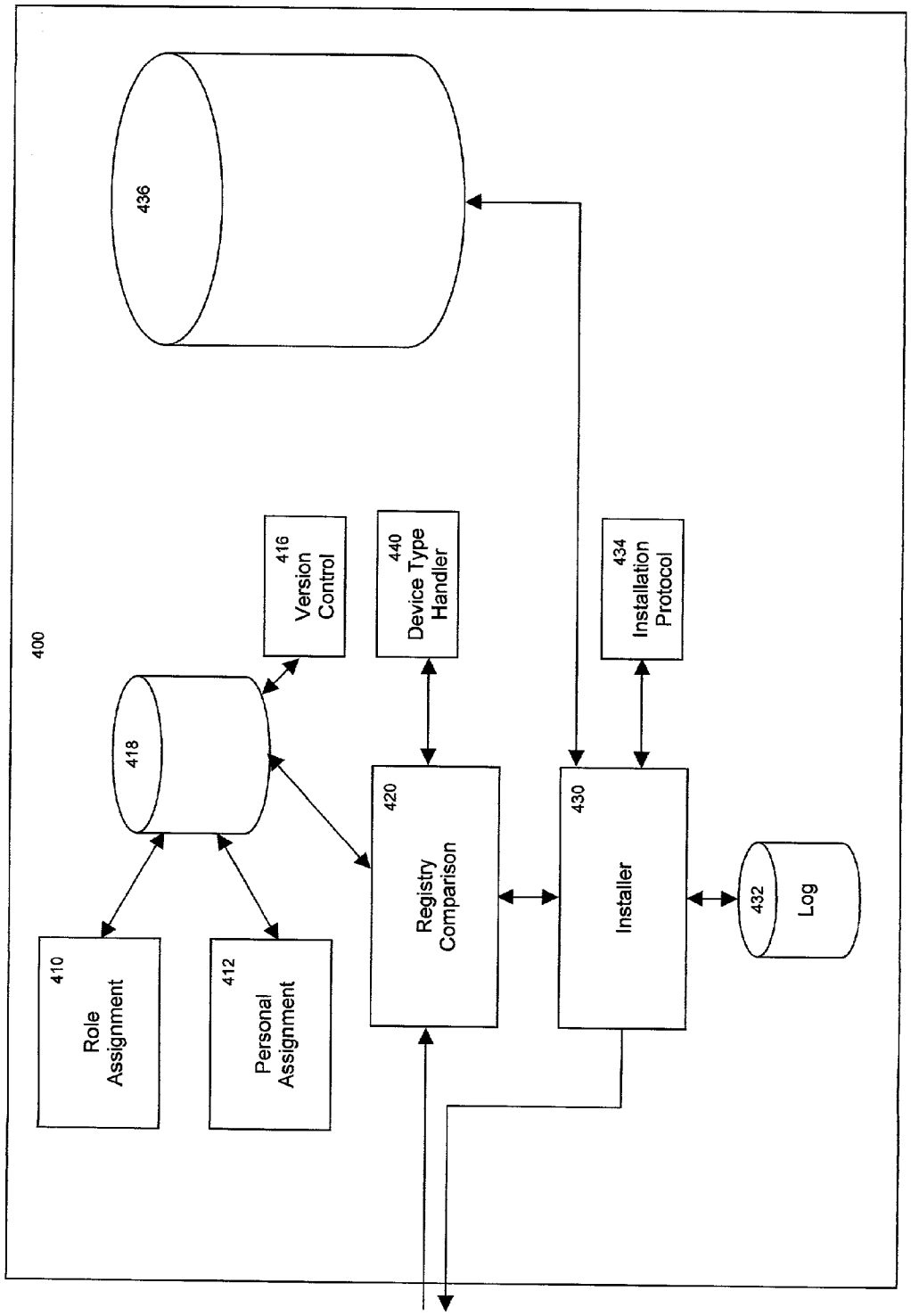

400
436
416
Version Control
440
Device Type Handler
434
Installation Protocol
418
420
Registry Comparison
430
Installer
432
Log
410
Role Assignment
412
Personal Assignment

Fig. 6

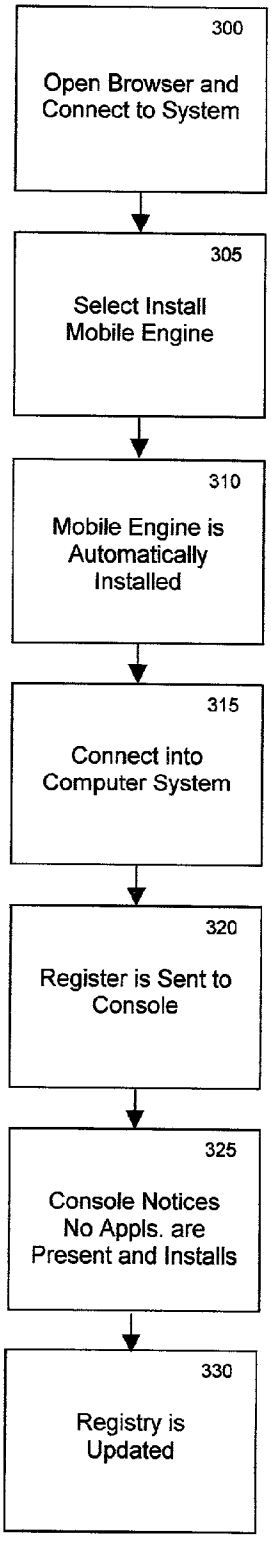

300
Open Browser and Connect to System
305
Select Install Mobile Engine
310
Mobile Engine is Automatically Installed
315
Connect into Computer System
320
Register is Sent to Console
325
Console Notices No Appls. are Present and Installs
330
Registry is Updated

Fig. 7

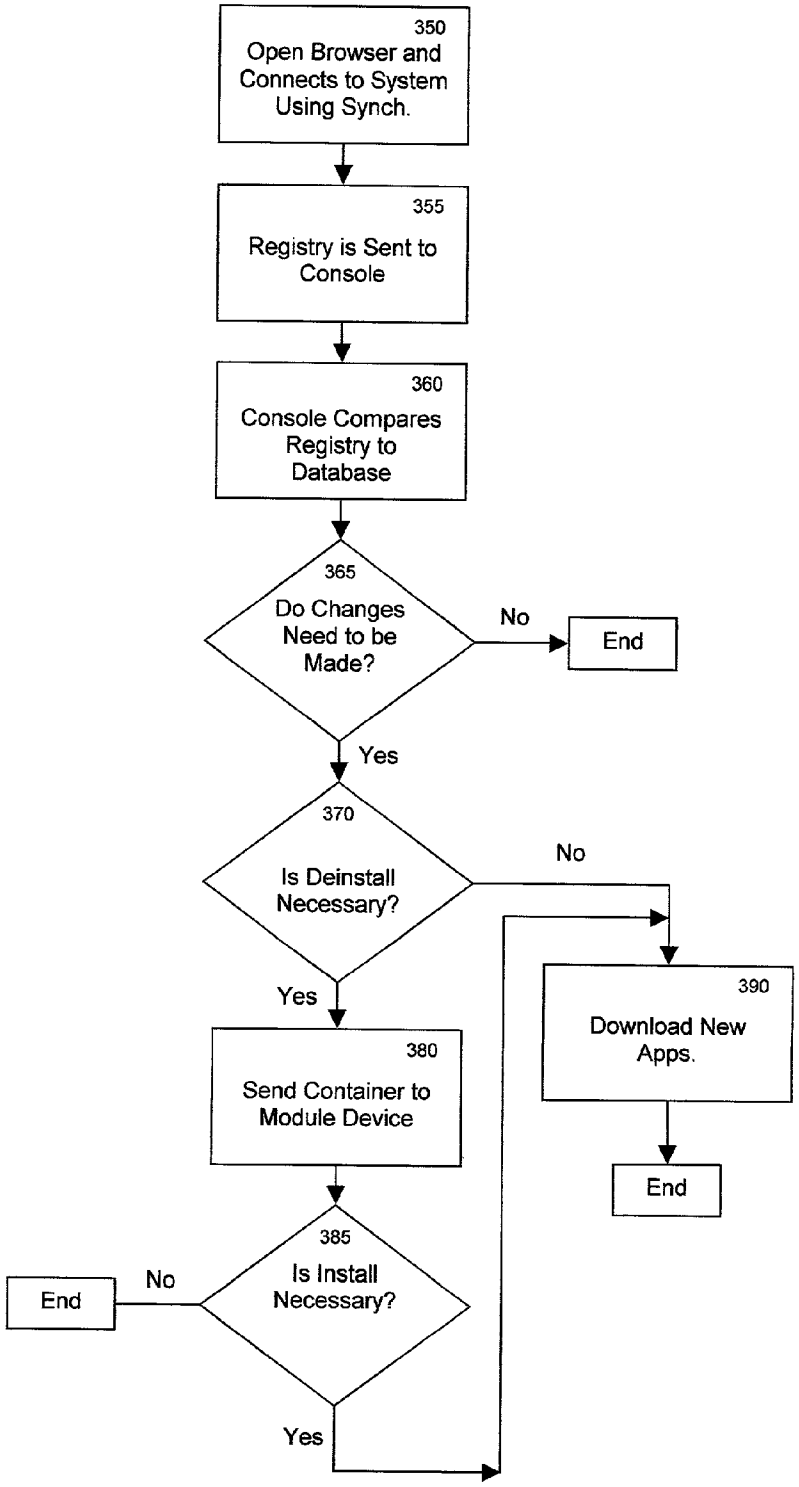

350
Open Browser and Connects to System Using Synch.
355
Registry is Sent to Console
360
Console Compares Registry to Database
365
Do Changes Need to be Made?
No
End
Yes
370
Is Deinstall Necessary?
No
Yes
380
Send Container to Module Device
385
Is Install Necessary?
End
No
Yes
390
Download New Apps.
End

Fig. 8

PROCESS FOR DEPLOYING SOFTWARE FROM A CENTRAL COMPUTER SYSTEM TO REMOTELY LOCATED DEVICES

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application arises from provisional applications No. 60/296,234 filed on Jun. 6, 2001 and 60/296,993 filed on Jun. 8, 2001 and claims the benefit thereof.

FIELD OF THE INVENTION

[0002] This invention relates to providing a process for deployment and installation of software on mobile devices, such as handheld computers, Internet capable phone, personal digital assistants (PDAs), and laptops, as well as desktop computers from a central computer system, so that such devices and computers may run business applications offline on a platform-independent framework and synchronize data with a computer system, such as a business electronic commerce system, via a standard Internet connection or other network connection.

BACKGROUND OF THE INVENTION

[0003] Many businesses have a need to exchange information between a central location and employees operating in the field. In the computerized age, most of this information is probably stored on a computer system. Because many field employees may have a laptop computer, handheld computer or personal digital assistant with which to work. Thus, it would be beneficial to enable the exchange of information between the computer system and the devices deployed in the field.

[0004] One example of a use for such a system would be a field engineer that responds to trouble calls. At the start of a day, the engineer may need to determine his daily tasks and at the end of the day he may need to report it back to his employer. Rather than get this information verbally, it would make more sense to download the tasks he has to perform that day from a central computer system. When he finishes the day, he would respond to the computer system with reports relating to the calls, setting forth the trouble found, the repair performed and the time it took to complete the job.

[0005] Another example may be a sales person who is calling on a customer. If that customer wants to change a previously placed order, the sales person may need to record the modified order and place that order for the customer. Doing this through a computer system would make much more sense than doing it verbally and would provide a record of the transaction.

[0006] Certain companies that have a need for providing devices like this may have certain employees that need more processing power than others. Thus, it may be necessary to deploy more than one type of mobile device. Some people may end up using personal digital assistants, while others use handheld computers, while still others use laptops. Others, that do not travel, may use desktops. For ease of use and management, it then becomes important that the software being used to perform the various tasks be platform-independent so that it may run on all these devices.

[0007] It also is important to permit the information to be entered on the devices while the device is off-line. Field personnel are not always in locations where they can establish connectivity to the computer system. It would be beneficial to permit the field personnel to enter the information they need to transmit back to the computer system when it is convenient for them, such as immediately following the repair, rather than making them enter the information while the device is on-line with the computer system.

[0008] It is also important for ease of use that when people are working off-line, the software operates in a similar way with the same look and feel as though they were actively operating on-line.

[0009] Because computer systems of this type can have hundreds or thousands of mobile device users that access the computer system from time-to-time, an easy way to keep track of what software is loaded on what device and to control the same is imperative. Thus, an easy way to control, manage, and monitor software installation on the devices is needed.

[0010] A need exists for a deployment and installation process for deploying and installing a platform-independent mobile framework and offline applications on mobile devices or desktop computers so that they may run business applications offline and synchronize data with a computer systems over a standard Internet connection or other network.

SUMMARY OF THE INVENTION

[0011] An embodiment of the present invention provides a process for deploying and installing a mobile framework on a device from a central computer system.

[0012] Another embodiment of the present invention provides a process for deploying and installing offline applications to run on a mobile framework on a device from a central computer system.

[0013] As such, it is an object of the present invention to facilitate the deployment and installation of a mobile framework on a device from a central computer system.

[0014] It is a further object of the present invention to facilitate the deployment and installation of offline applications to run on a device from a central computer system.

BRIEF DESCRIPTION OF THE DRAWINGS

[0015] **FIG. 1** is a block diagram of a mobile device according to an embodiment of the present invention.

[0016] **FIG. 2** is a block diagram of a mobile device having an API layer according to an embodiment of the present invention.

[0017] **FIG. 3** is a block diagram of a synchronization system according to an embodiment of the present invention.

[0018] **FIG. 4** is a block diagram of a computer system having a synchronization system according to an embodiment of the present invention.

[0019] **FIG. 5** is a flow chart illustrating a synchronization process according to an embodiment of the present invention.

[0020] **FIG. 6** is a block diagram of a deployment console according to an embodiment of the present invention.

[0021] **FIG. 7** is a flow chart illustrating a deployment process according to an embodiment of the present invention.

[0022] **FIG. 8** is a flow chart illustrating a deployment process according to another embodiment of the present invention.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0023] The present invention will be better understood by reference to the accompanying drawings.

[0024] **FIG. 1** depicts a mobile device according to an embodiment of the present invention. Mobile device **100** may be a PDA, a laptop computer, an Internet capable phone, or another similar device. Preferably, the mobile device has an operating system with Java VM 1.1.4 or PersonalJava and at least about 5 MB of memory. Preferably, mobile engine **110** is based on Java, so that it may run on any platform supporting Java.

[0025] Mobile device **100** contains an Internet browser **105**. It also contains mobile engine **110**. Mobile engine **110** should be capable of generating any standard markup language, such as HTML, so that Internet browser **105** may be a standard browser, such as Microsoft Internet Explorer or Netscape, rather than a proprietary one. However, a proprietary browser could be used on mobile device **100** in addition to or in lieu of the standard browser if desired.

[0026] When mobile device **100** is operating with a computer system over the Internet in an online mode and applications are being run from computer system **200** across the connection, browser **105** is connected to computer system **200**, such as an application server, in a traditional manner represented by box **197**.

[0027] When mobile device **100** is operating in an offline mode and applications are being run locally on mobile device **100**, mobile engine **110** is engaged. Mobile engine **110** contains two major components: a Java plug-in **90** and various components for data synchronization and deployment. The Java plug-in contains web server **115**, servlet engine **120**, modular offline application building blocks **125**, **130** and **135**, and API layer **145**. All of the information to start and run the Java plug-in is preferably held in separate property files. For instance, the information related to the web server, including host, port, and wwwroot, can exist in a file named webserver.properties. The information related to mappings, such as URL to servlet mappings, can exist in a file named mappings.properties. The information relating to mimes, such as mime types, can exist in a file named mime.properties.

[0028] Web server **115** is preferably programmed in Java and provides the ability for the user of mobile device **100** to log onto web server **115** contained within mobile device **100** when operating offline. By doing so, the user is provided with the same or similar look and feel to operating online over the Internet and can run the same applications offline that the user could run over the Internet in an online mode. Preferably, web server **115** is single threaded and has a small footprint.

[0029] Web server **115** is connected to servlet engine **120**. Servlet engine **120** enables the mobile device's user to engage various offline applications locally on mobile device **100** through web server **115**. Servlet engine would preferably be based upon Java Servlet API 2.1. Web server **115** and servlet engine **120** preferably would meet the Sun Java WebServer 2.0 Specification.

[0030] Various different programming models can be deployed in mobile engine **110** as modular offline building blocks. For instance, a proprietary model 125, like SAP-Market's MicroITS, may be deployed which is designed to maximize certain types of transactions based upon the foreseen use of mobile device **100**. MicroITS model 125 contains a flow logic parser, HTML business parser, language resource parser/preparser and module provider.

[0031] A Java Server Pages model 130 may be deployed to take advantage of Java features. Java Server Pages model 130 would preferably be based upon TomCat.

[0032] Other models, such as custom plug-in model 135, can also be used as needed. Custom plug-in model would permit users to implement their own logic. These other models may be proprietary or standards-based. By using modular offline application building blocks and permitting the use of one or more models, the mobile engine architecture is easy to enhance or modify. Moreover, new offline application building blocks can be downloaded and installed via the deployment and installation process discussed hereinafter with respect to **FIGS. 6 and 7**. Mobile engine API layer **145**, which will be discussed in more detail below, provides independence between modular offline application building blocks **125**, **130**, **135** and servlet engine **120**.

[0033] Modular offline application building blocks **125**, **130**, and **135** utilize programming sources stored in memory **140**. While memory **140** is shown within mobile engine **110**, it may be located elsewhere.

[0034] Mobile engine API layer **145** forms an interface between the various programming models **125**, **130** and **135** and the application data that resides on database **180** within mobile device **100**. While database **180** is shown within mobile engine **110**, it may be located elsewhere. By providing API layer **145**, alternative programming models, offline applications and services may be incorporated, activated or substituted for existing models, offline applications and services in the future without having to completely reprogram mobile engine **110**. API layer **145** contains file I/O APIs, database APIs, synchronization layer APIs and XML parsing APIs.

[0035] This mobile engine API layer **145** also provides device independence so that any application can run on any device without specific programming for device specific dependencies. Referring now to **FIG. 2**, API layer **145** provides an interface between application or service **50** and the mobile device's file programming interface **40** on the mobile device platform **45**. Service **50** may be any of the previously mentioned services, such as data exchange **150**, or the like, or an offline application.

[0036] API layer **145** consists of various module providers, such as EBP.class **5**, ISALES.class **10** and MY.class **15**. Java methods grouped in packages **20**, **30**, and **35**, such that all methods needed for a specific function are part of a corresponding package, also form part of API layer **145**. These methods can be called by the appropriate module provider.

[0037] For instance, Enterprise Buyer Professional is a product available from SAP Markets, Inc. It provides the ability to coordinate the Internet business procurement process. The product permits the creation of shopping baskets

and catalogs. A package for Enterprise Buyer Professional would include the module provider of EBP.class **5** and the groups of methods basket.class **20***a* and catalog.class **20***b*.

[**0038**] Module providers **5**, **10**, and **15** instantiate the appropriate Java method calls and make them available for the flow logic in an application or service. There is always one module provider assigned to each package of the API layer **145**.

[**0039**] Various services that can be integrated that operate through API layer **145** are shown. Data exchange **150** is utilized during a synchronization procedure when data is to be exchanged between a computer system and mobile device **100**.

[**0040**] Many business applications utilize XML. So an XML translation service **155** can be provided.

[**0041**] A personalization service **160** can be included. Personalization permits the manipulation of settings necessary to operate within a given server environment, such as a server URL, logon data, proxy, etc. Multiple user control can be included if more than one user of a device would be anticipated.

[**0042**] An installation service **165** can also be provided. Installation service **165** automatically installs or uninstalls offline applications, offline application building blocks and other software so that the device is outfitted as it should be based upon a deployment console, as will be discussed in detail later, located at the computer system. The installation service **165** creates a new entry in registry service **175** (discussed below) when a new application is installed and deletes an entry when an offline application is deleted. This function occurs in the background so as to be transparent to the user.

[**0043**] Local database access **170** is provided to permit access to database **180**.

[**0044**] A home service **172** can be provided that sets up a homepage that the user accesses offline that provides links to the offline applications resident on the mobile device. Preferably, the synchronization service is accessible through this page by clicking on an icon, for example.

[**0045**] Library service **174** offers standard functionality for application methods. It handles file operations and parsing. Library service **174** contains an open catalog interface and parser for XML parsing, as well as a local data storage encryptor.

[**0046**] Registry service **175** maintains a list of all installed offline applications. Alternatively, registry service **175** can maintain a list of all installed software subject to deployment from the deployment console, which would include at least offline applications and modular offline application building blocks. Preferably, information from the registry **175** is provided to the deployment console during a synchronization and used by the deployment console to make sure that mobile device **100** is outfitted as it should be. The operation of registry service **175** will be discussed in more detail later.

[**0047**] Synchronization layer **185** is part of mobile engine **110**. Synchronization layer **185** controls the synchronization of data on mobile device **100** with computer system **200** once the mobile device achieves connection through the Internet (or alternatively, through another means) to computer system **200**, such as an application server, as is represented by box **195**. The synchronization layer **185** does this by sending the data containers resident in synchronization folder **187**. Synchronization layer **185** preferably will contain an inbound/outbound queue, module(s) supporting different types of synchronization, a file handler, an error handler, a SOAP connector for XML, a data transmission security module, a synchronization security module and an authority checker. The details of the creation of such components would be apparent to one skilled in the art. This synchronization layer **185** and synchronization process will be discussed in greater detail later.

[**0048**] A few examples of offline applications that can be run on mobile device **100** are now discussed. Such applications can be created through the use of a tool such as Web Application Builder by SAP, AG. One such application is an easy shopping application. Easy shopping permits one-step wireless shopping and buying, personalized product offerings, seamless integration into a full internet sales cycle, seamless integration into a procurement cycle, intelligent status tracking/synchronization, easy changing web templates, XML catalog content exchange, and provides the identical look and feel for sales and procurement. Functions supported could be catalog, shopping cart, synchronization, and order status. The catalog can include search functionality and can hierarchically group products. The shopping basket allows for the creation of multiple orders and provides order status information. When this data from this application is synchronized with the central computer system, catalog content, orders, and software can be updated.

[**0049**] Manager's inbox is another possible offline application. Manager's inbox contains an inbox overview listing incoming messages and work items. View selection can be available. When an inbox item is clicked, the message or the work item can be displayed. Work item details such as the ability to approve or reject a work item and forms can be shown. During synchronization, inbox items, item details, and approvals/forms would be updated.

[**0050**] Plant maintenance is another application. Plant maintenance includes an order section. Functions possible with plant maintenance could include searching for open or released orders, selecting operations, and adding components. The catalog function can provide a hierarchical view catalog with search functionality. During synchronization open and released orders would be exchanged, catalogs would be updated and new components would be added.

[**0051**] The discussion above, with respect to **FIG. 1**, focused on the use of the framework with a mobile device. Another use for this framework would be to install it as described above on a desktop computer rather than a mobile device or on a laptop that has a more or less permanent network connection. Having such a framework would permit users to run applications offline and then exchange information between the desktop and the computer system during a synchronization process. Thus, when offline application data needs to be provided to the computer system, it is provided to the system through the synchronization layer. The synchronization process can occur upon the clicking of an icon or hyperlink or the like, or alternatively, because a network connection does exist, it can automatically be provided without user intervention when data is available to be sent.

[**0052**] By using the mobile engine framework on a desktop, or laptop with a somewhat permanent network connection, the volume of exchange of information between the computer system and the desktop can be minimized. This would mean that entire HTML pages would no longer need to be exchanged. Instead, just the data of the business objects would be exchanged.

[0053] In **FIG. 3, a** synchronization system according to an embodiment of the present invention is shown. Mobile device **100** is shown connected to computer system **200** through a connection such as the Internet. Other networks or a direct connection can alternatively be used. The synchronization process could begin through clicking on an icon, button, hyperlink or the like, on a home page displayed via the browser. Alternatively, it could automatically occur upon a link being established between mobile device **100** and computer system **200**.

[0054] Sample Java code for initiating a synchronization process follows:

```
try{
    SyncOutboundContainer.doSync();
} catch (SyncException ex) {
    // do something
}
```

[0055] Mobile engine **110**, through API layer **145** and synchronization layer **185**, talks with functional module **220** on computer system **200** through synchronization RFC **210**. This communication is carried out through the use of data containers, such as data container **230**.

[0056] Data container **230** is normally made up of a header **240** and a body **250**. However, certain types of containers may not require a body **250**. The header **240** can be made up of the following parts: 1) a container ID that uniquely identifies the particular container; 2) an identification of the user of the device; 3) an identification of the type of container; 4) the method describing which function module should be called in the computer system to process the data; 5) the date the container is sent; 6) the time the container is sent; 7) the date the method was executed to create the data container; 8) the time the method was executed; and 9) the status relating to the data. The body **250** of container **230** can be made up of a container ID, line number, field name and field value.

[0057] The following is sample Java code for an outgoing data container:

```
String MyOwner; // best the username on the PDA
try{
    SyncOutboundContainer syncOutboundContainer = new
SyncOutboundContainer(MyOwner,GETAPPROVERLIST );
    syncOutboundContainer.open();
    String value_for_I_CFGUID = // do something to fill
value
    syncOutboundcontainer.addItem("I_CFGUID", 0,
value_for_I_CFGUID);
    String value_for_I_OBJTYPE = // do something to fill
value
    syncOutboundContainer.addItem("I_OBJTYPE", 0,
value_for_I_OBJTYPE);
    int NoOfLinesInTable_E_APPROVER;
    for (int line=0; line < NoOfLinesInTable_E_APPROVER ;
line++) {
        String line_for_E_APPROVER = // do something to fill
line
syncOutboundContainer.addItem("E_APPROVER",line,line_for_E_
APPROVER);
    }
    syncOutboundContainer.close();
} catch (SyncException ex) {
    // will always happen if method "GETAPPROVERLIST " is not
registered
}
```

[0058] **FIG. 4** shows a computer system with synchronization capability and **FIG. 5** shows a synchronization process according to an embodiment of the present invention. Preferably, the synchronization layer on the computer system has an inbound/outbound queue, module(s) for different types of synchronization, a dispatcher, a spooler, a file handler, an error handler, a mapping algorithm, a data transmission security module, a synchronization security module, and an authority checker. The details of the creation of such components would be apparent to one skilled in the art.

[0059] The embodiment shown in **FIG. 5** includes an optional data preservation scheme. In order to preserve the containers, a container is maintained on at least one location. The container on the mobile device will remain on the mobile device if a problem occurs such as the connection getting lost during synchronization. If that occurs, it will be sent automatically with the next synchronization. If there is no entry in the mapping table for the function module call, the container remains in the incoming table on the computer system and a program is used to generate an automatic mapping, as will be discussed later.

[0060] If the called function module aborts and creates a dump, the container remains in the incoming table. If the called function module has the wrong interface implemented, it remains on the incoming table. If the data contained in the container is errored, the called function module returns an error message to the mobile device for processing by the offline application.

[0061] Making reference now to **FIGS. 4 and 5**, in step **230** of **FIG. 5**, the user of mobile device **100** clicks the synch icon, button, hyperlink, or the like, on the home page to initiate the synchronization process. The synchronization class reads all data containers in the synchronization folder and compares the containers with a history table that records the container IDs for containers already sent by the mobile device **100** and received by computer system **200** as shown in step **232**. This is done so that duplicate containers, should they arise for some reason, such as a reboot of the mobile device, can be deleted from the mobile device without sending them.

[0062] In step **234**, the synchronization class sends the remaining containers over the synchronization transport layer to synchronization RFC function module **210** in the computer system in step **252**. In step **236**, the synchronization RFC function module **210** writes the incoming containers into container database **260**. The incoming containers are stored in database **260** as a failsafe to ensure that the containers are available should a problem occur after they are passed on to mapping layer **265**. Synchronization RFC function also reads out from the outgoing container database **275** any data containers with the same class and user as those received and sends them to mobile device **100**. Additionally, it sends an acknowledgement for the containers just received and stored in incoming container database **260**.

[0063] In step **238**, the mobile device **100** then writes the container ID referred to by the acknowledgment into a history table indicating that it was previously received by the computer system **200** and no longer needs to be maintained on the mobile device **100**. The container is then deleted off of mobile device **100**.

[0064] In step **240**, the incoming containers are read out of inbound container database **260** by scheduled function mod-

ule **265**. In step **242**, scheduled function module **265** consults mapping table stored in database **270** to map the data within the container so that it may be processed accordingly. Scheduled function module **265** then calls the appropriate function module **220** to execute the function required in step **244**. One such function module could be an EBP processing function, for example.

[**0065**] In step **246**, appropriate function module **220** sends back responsive data, such as status, to scheduled function module **265**. Scheduled function module **265** then places the data in a container and writes it to outgoing container database **275** in step **248**. Depending on the type of incoming containers, the responsive outgoing containers may be sent immediately in step **249** or may be stored until the next time a synchronization process is initiated.

[**0066**] At least three separate types of synchronization would preferably be supported. The first type is publish synchronization. In publish synchronization, the type of container used can be called an outbound container. The outbound container is created in mobile device **100** and, when a connection is present, is sent to computer system **200** for processing. Once it has been received by computer system **200**, any return containers waiting to be sent to the same user and of the same method are sent back to the user. The connection is terminated however, prior to any outgoing containers responsive to the data just sent by the user being sent back to the mobile device.

[**0067**] The second type of synchronization is online processing. This type is similar to publish but includes return of responsive outgoing containers having processed data relating to the container that was just sent. With online processing synchronization, the owner sends a request-type container. First the connection is made. Then the request-type container is sent. The container is then received by the computer system **200**. Because it is a request type container, the computer system returns not only any containers waiting to be sent to the same owner of the same method, but also processes the incoming container and sends a response to that container prior to the connection being terminated.

[**0068**] The third type of synchronization is subscribe. Subscribe synchronization is used to check on the status of previously sent containers. For instance, it can be used to check on the status of an electronic order placed through a shopping cart. The owner sends a notify-type container to computer system **200**. Computer system **200** then returns containers of same method and user. Such containers need not include a body, as the status information is contained within the head of the containers.

[**0069**] Development tools can be utilized with the mobile engine according to an embodiment of the present invention. On the computer system, a generator tool can generate the wrapper function module that maps the generic structure of a container to the individual interface of a function module and generate the table entries for the mapping table. Preferably, the tool will utilize the programming language of the computer system. A tool can also be used as a Java class generator to handle flow logic to Java for modular offline application building blocks **125** and/or **135**. Preferably, the tool utilizes Java on the mobile device. These tools are essentially translators. Creation of such translators would be well within the abilities of one of ordinary skill in the art. A computer aided test tool can also be created and used for modular application building blocks **125** and/or **135**.

[**0070**] Referring now to **FIG. 6**, to handle the management of the myriad of differently programmed mobile devices, a deployment console **400** is resident on the computer system as one of the application function modules **220**. Deployment console **400** is preferably programmed in a language supported by computer system **200**. For example, if the computer system is a MySAP Workplace system by SAP, AG, then the deployment console could be programmed in ABAP.

[**0071**] Deployment console **400** has an overview of all installed offline applications per user and device through an installation log **432** that records that an installation has occurred whenever an offline application, modular offline application building blocks or other software is deployed and installed on a device.

[**0072**] Some computer systems serving a plurality of users install applications according to user roles. For example, an employee may have a role of salesman, field engineer, secretary, or the like. Role assignment module **410** permits an administrator to assign a user to a role and to assign what software should be installed for each role. A user may have one or more roles if desired.

[**0073**] Because, in some cases, deviation from a strict role-based regiment may be desired, personal assignment module **412** can be provided. Personal assignment module **412** would permit an administration to assign an application that may be needed by an individual, but not by others within the same role. For example, a vice president of sales may be assigned a role of vice president, but may need some but not all applications of the salesman role. Personal assignment module **412** would permit such customization.

[**0074**] The deployment console **400** can also manage versions of software for deployment by defining the current version of each offline application that should be installed. Version control module **416** provides an administrator with this functionality.

[**0075**] Storage media, such as database **418**, keeps track of what software should be installed on each device for each user. This data stored therein is based upon the parameters set by role assignment module **410**, personal assignment module **412** and version control module **416**.

[**0076**] Storage media **436** is provided upon which software that may be downloaded to a device and installed thereupon is provided. Such software may include offline applications, modular offline application building blocks, patches, and other software.

[**0077**] Installation protocol module **434** is provided. It permits an administrator to update or alter installation protocol.

[**0078**] Device type/ID handler **440** is provided which determines how to handle different device types.

[**0079**] Installer **430** is provided that retrieves software from storage media **436** for downloading and installation on a device. This is done based upon the comparison of information from a registry service on a device to information stored in database **418** relating to what is supposed to be resident on that particular device. Additionally, installer **430** is equipped to provide mobile device with a deinstall instruction should the information from the registry service indicate that software is installed on the device that should not be.

[0080] When installer **430** retrieves software from storage media **436** and sends it off to a device and retrieves a notice from the device that the software was received, it stores the information relating to the download and install in install log **432**. Install log **432**, database **418**, and storage media **436** may be collocated or separate. Moreover, they may be a part of deployment console **400** as shown, or alternatively, any of them may be located externally.

[0081] A process for deploying and installing the framework according to an embodiment of the present invention is shown in **FIG. 7**.

[0082] In order to install the mobile engine framework, a user opens browser **105** on mobile device **100** and connects to computer system **200** online as shown in step **300**. Once online, an install mobile engine icon, hyperlink or the like is displayed. In step **305**, the user selects to install mobile engine **110** by clicking on the icon, hyperlink or the like.

[0083] In step **310**, the mobile engine framework is downloaded to mobile device **100** and, preferably, is automatically installed by installation service **165**. Once installation is complete, the user connects into the deployment console system in step **315**.

[0084] Upon this connection, a data container containing information from registry service **175** is sent to a deployment console on computer system **200**. As shown in step **325**, the console reads the registry and determines that no offline applications are currently installed on mobile device **100** and then downloads the appropriate applications that are supposed to reside on mobile device **100**. These offline applications are installed automatically by installation service **165**. Installation service **165** also automatically updates registry service **175** to reflect the installation of the new offline applications.

[0085] When a deployment console receives information from a registry service that indicates an outdated version of a program or a version requiring a patch is resident on a mobile device, the current version and/or patch can be sent to the device and installed and the old version deinstalled, as needed, just as any other program, as will be described with respect to **FIG. 8**. It should also maintain error logs sent by the mobile devices.

[0086] In **FIG. 8, a** process using the registry in association with the deployment console is depicted. In step **350**, the user opens browser **105** and connects to computer system **200** through the Internet, or the like, using the synchronization function. As part of the synchronization process, mobile device **100** then sends a data container containing the data within the registry service **175** that is eventually retrieved by the deployment console as shown in step **355**. This retrieval may be in the manner discussed above relating to the retrieval of data from data containers during the synchronization process.

[0087] The console then compares the data from registry service **175** to stored information that reflects what offline applications (or alternatively what software that is subject to deployment through deployment console) are supposed to be resident on mobile device **100** in step **360**. In step **365**, it is determined if changes need to be made to the mobile device. A few examples of changes that may need to be made include: 1) an offline application may no longer be authorized and may need to be deleted from the mobile device; 2) a new application may need to be added; 3) a new version of an existing application may need to be placed on the mobile device; 4) a patch for an existing application may need to be installed on the mobile device; 5) modular application building blocks may need to be deleted and/or installed; and 6) other software such as special extra HTML pages and/or graphics, for example, may need to be added or deleted, such as for a "Christmas" special.

[0088] If no changes need to be made, then the routine ends. If changes do need to be made, it is determined if a deinstall or an install needs to be performed. In step **370**, it is determined if a deinstall is needed. If one is, then a data container is sent to mobile device **100** directing installation function **165** to deinstall the appropriate offline application or other software. It is then determined if an install is necessary in step **385**. If no install is needed, then the routine ends. If an install is needed, in step **390**, the offline application or other software to be installed is downloaded to mobile device and installed automatically through the installation function **165**.

[0089] This sending of the registry occurs each time there is a synchronization process. This way the deployment console keeps the mobile devices outfitted as they should be.

[0090] Although the preferred embodiments of the present invention have been described and illustrated in detail, it will be evident to those skilled in the art that various modifications and changes may be made thereto without departing from the spirit and scope of the invention as set forth in the appended claims and equivalents thereof.

What is claimed is:

1. A deployment process for a device facilitating deployment and installation of a framework from a computer system to said device, comprising the steps of:

opening a browser;

connecting said device to said computer system;

sending an indication to said computer system that said framework is to be installed on said device;

receiving said framework from said computer system;

installing said framework;

sending a registry to said computer system, said registry comprising information indicative of a lack of offline applications being installed on said device;

receiving offline applications from said computer system;

installing said offline applications.

2. A deployment process as in claim 1, wherein said installation steps occur automatically.

3. A deployment process as in claim 1, wherein said device comprises a personal digital assistant.

4. A deployment process as in claim 1, wherein said device comprises a handheld computer.

5. A deployment process as in claim 1, wherein said device comprises a laptop computer.

6. A deployment process as in claim 1, wherein said device comprises a desktop computer.

7. A deployment process as in claim 1, wherein said device comprises an Internet-enabled phone.

8. A deployment process as in claim 1, wherein said computer system comprises an applications server.

9. A deployment process for a device for deploying offline applications from a computer system to said device, comprising the steps of:

opening a browser;

connecting said device to said computer system;

sending a registry to said computer system, said registry comprising an indication of each offline application installed on said device;

if an offline application is to be deleted from said device, receiving a message from said computer system to deinstall said offline application to be deleted;

if a new offline application is to be installed on said receiving said new offline application from said computer system and installing said new offline application.

10. A deployment process as in claim 9, further comprising a step of updating said registry upon an installation or deinstallation.

11. A deployment process as in claim 9, wherein said deinstallation takes place automatically.

12. A deployment process as in claim 9, wherein said installation takes place automatically.

13. A deployment process as in claim 9, wherein said device comprises a personal digital assistant.

14. A deployment process as in claim 9, wherein said device comprises a handheld computer.

15. A deployment process as in claim 9, wherein said device comprises a laptop computer.

16. A deployment process as in claim 9, wherein said device comprises a desktop computer.

17. A deployment process as in claim 9, wherein said device comprises an Internet-enabled phone.

18. A deployment process as in claim 9, wherein said computer system comprises an applications server.

19. A deployment process for a computer system for facilitating deployment of a framework from said computer system to a device, comprising the steps of:

receiving a notification that said framework is to be installed on said device;

downloading said framework to said device;

receiving a registry from said device, said registry comprising information indicative of a lack of offline applications being installed on said device;

reading said registry;

determining which offline applications should be installed on said device;

downloading said offline applications that should be installed on said device to said device for installation.

20. A deployment process as in claim 19, wherein said device comprises a personal digital assistant.

21. A deployment process as in claim 19, wherein said device comprises a handheld computer.

22. A deployment process as in claim 19, wherein said device comprises a laptop computer.

23. A deployment process as in claim 19, wherein said device comprises a desktop computer.

24. A deployment process as in claim 19, wherein said device comprises an Internet-enabled phone.

25. A deployment process as in claim 19, wherein said computer system comprises an applications server.

26. A deployment process for a computer system for deploying offline applications from said computer system to a device, comprising the steps of:

receiving a registry from said device, said registry comprising an indication of each offline application installed on said device;

determining which of a plurality of offline applications are intended to be installed on said device;

comparing said registry to said determination to devise if a discrepancy exists;

if said discrepancy exits and an offline application on said device is not intended to be installed, sending a deinstall message to said device, said message comprising an instruction to deinstall said offline application not intended to be installed;

if said discrepancy exists and an offline application intended to be installed on said device is not installed on said device, downloading said offline application intended to be installed to said device for installation.

27. A deployment process as in claim 26, wherein said device comprises a personal digital assistant.

28. A deployment process as in claim 26, wherein said device comprises a handheld computer.

29. A deployment process as in claim 26, wherein said device comprises a laptop computer.

30. A deployment process as in claim 26, wherein said device comprises a desktop computer.

31. A deployment process as in claim 26, wherein said device comprises an Internet-enabled phone.

32. A deployment process as in claim 26, wherein said computer system comprises an applications server.

* * * * *