Ubiq Mobile – a new universal platform for mobile online services

Valentin Onossovski, Andrey N.Terekhov

St.-Petersburg State University, Faculty of Mathematics and Mechanics
Department of Software Engineering
198504, Universitetsky av.,28, St.-Petersburg, Russia
v.onossovski@gmail.com, ant@tercom.ru

Abstract

The paper describes basic ideas, architectural structure and details of implementation of the original universal platform – called Ubiq Mobile – that has been developed by the team of software engineers from St.Petersburg State University for creation of mobile online services. The platform combines extended functionality and rich user experience with traffic and resource optimization that lets mobile services work efficiently in comparatively slow networks and on a wide set of mobile handsets. The principal novelty of our approach is that the mobile device is considered as a terminal rather than an active client, and all business logic of the mobile application is implemented on the server side. For providing traffic and resource optimization, many sophisticated mathematical methods and algorithms are implemented in the platform. Data exchange between server and mobile device is performed through a custom binary protocol that provides high speed, reactivity and mobile traffic optimization as well as rich functionality.

Index Terms: mobile online services, universal platform, mobile terminal, custom binary protocol

I. Introduction

The process of convergence between Internet and mobile technologies is getting an important trend in the field of popular Internet services. Most of popular online services have their mobile versions; such industry giants as Google, Nokia, Sun Microsystems and other large companies express an outstanding interest to this area. At the same time, mobile online services are not as prevalent in our day-to-day life as we could expect, notwithstanding an impetuous progress and growing abilities of modern mobile devices.

We believe that one of the main reasons that prevent Internet services from quick migration from PCs onto mobile devices is that these two categories of devices fundamentally differ in terms of their use. On the one hand, mobile devices have smaller screen size, specific input abilities (small keyboard or touch screen) and significant resource restrictions in comparison with traditional PCs. On the other hand, degree of "personal attachment" of a mobile device to its owner is much higher than of a PC – usually people bring their handsets with them permanently and don't pass them to the others (that gives additional possibilities, in particular, for implementing authentication procedures and location-based services). It means that the straightforward approach based on just porting client parts of existing online services from PC to mobile devices is not satisfactory and that for better usability mobile online services (especially their UI) should be designed specifically for handheld devices, taking into consideration all their peculiarities.

Among different types of mobile online applications, services that use service-specific "fat" mobile clients combining rich user experience, high efficiency and resources optimization show the

best results. The web-based mobile applications (running inside Web browser) either provide too poor functionality (like WAP-based), or, if using advanced Web technologies, consume sufficient resources – computational power, broadband wireless connection for mobile traffic etc.

Unfortunately, the development of "fat" clients is rather expensive due to the effect of platform fragmentation (differences in implementation of the same language or platform standard – like J2ME – on different mobile devices). So, only large and powerful companies (like Google) can afford to develop "fat" client programs for their mass mobile services and support them for wide range of mobile devices. For many independent developers of mobile services the only suitable way is to look for some universal platform that would encapsulate most of device-dependent specifics and would produce code, possible to run on different mobile phones. Nokia widgets, mobile Ajax technology [2] and Sun MSOA [3] are striking examples of such class of universal platforms that utilize different ideas and approaches.

The team of software engineers from St.Petersburg State University has developed a new platform – Ubiq Mobile – that encapsulates most of mobile device-dependent technical details and lets programmers develop modern, functionally rich and highly interactive mobile online services that can run on different mobile devices. Ubiq Mobile platform provides high efficiency and sufficient traffic and resource optimization – so that the mobile services based on this platform can work properly slow mobile data connections (like GPRS and EDGE) and on wide range of mobile devices.

II. UBIQ MOBILE PLATFORM OVERVIEW

The Ubiq Mobile platform provides the following set of features:

- High interactivity
- Rich user experience
- Efficient traffic optimization
- User sessions persistence
- Working on different types of mobile devices
- Support of positioning for mobile devices with embedded GPS
- Working in different conditions of mobile network connectivity (including relationally slow GPRS data transfer and unstable mobile connections)
- Easy programming of distributed mobile applications in various programming languages

The platform provides level of functionality and user experience comparable with the level of universal technologies (like Mobile AJAX) but, in contrast to them, is able to work efficiently with slower mobile connections and on wider set of mobile handsets, including relationally cheap mobile phones. Such frugality of our technology empowers developers to create mobile services with wider distribution among mass end-users than more traditional services based on existing "heavyweight" technologies.

Our approach is based on the following key ideas:

- "Mainframe-like" structure: mobile devices are considered to be graphic terminals rather than "traditional" clients. Applications run on the server side and perform graphical I/O via mobile devices

- Client component is unified for all types of supported mobile devices
- Data exchange between server and its mobile clients is mostly graphical; image data compression is widely used
- A proprietary binary protocol is built over TCP/IP
- Sophisticated mathematical algorithms for traffic and resources optimization are used
- Server software runs on Microsoft.NET platform that lets developers use different languages for programming mobile applications
- Client software supports Symbian S60, Windows Mobile and J2ME MIDP2 platforms

For our platform we have chosen terminal architecture instead of traditional client-server one because it allows to combine rich user experience with high interactivity and low traffic requirements. The obvious drawback of terminal architecture is its inability to provide offline work. But, from our point of view, it won't cause serious problems because we expect the platform to be primarily used for creation of interactive information services, where active offline work doesn't make much sense.

User applications run on the server side and mostly don't care about mobile specifics. From the application viewpoint, screen output operations are fully transparent – the application just draws graphics on virtual canvas in server's memory via platform API. The platform automatically supports correspondence between canvas and screen of mobile device via custom protocol. The device screen is considered to be a "window" that permits user to see either the whole canvas or some its part (if canvas size is bigger than screen size). The user can move this window along the canvas by using scrolling keys or joystick on the mobile device.

User input delivers to the application all information about various user events – pressed keys, selected screen controls, user input etc. The application handles such events via platform API either in synchronous or asynchronous mode.

The I/O model provided by the platform for user applications is mostly device-independent. The only exception is a mechanism of device-dependent functions invocation that is used, in particular, for getting information about mobile client location. User application can invoke device-specific function on mobile device. If this function is supported by particular device, the result of function invocation will be passed back to the server; otherwise, special "not supported" value will be returned to the application.

Graphical data exchange between server and mobile clients is performed via proprietary binary protocol built over TCP/IP. The main reason for using custom protocol instead of one of standard open protocols relates to traffic and resources optimization. The model of data exchange is sufficiently based on the knowledge of canvas logical structure and using custom protocol allows to minimize redundancy of passing data and optimize mobile traffic.

We can see that for many functionally rich online applications built on traditional Web technologies broadband data connection and sufficient processor resources are strictly necessary for their proper work. This is the main bottleneck preventing such applications from being really ubiquitous and highly interactive, especially when working with slow networks and on simple mobile phones. Using sophisticated mathematical algorithms, we tried to go through the eye of the needle and to pull down both these characteristics below some threshold level for providing stable and robust work of the system under any conditions. In many cases, we use specially optimized methods and algorithms instead of the standard ones – in particular, we use special modifications of image compression methods for graphical data exchange between server and mobile clients. The

platform includes a mechanism of "on-air tuning" that constantly adjusts key performance parameters (such as size of binary packets passed to the mobile client) depending on current network conditions.

To make development of mobile applications easy for programmers, we implemented server part of Ubiq Mobile platform in Microsoft .NET environment. User applications are just .NET components that can be deployed to the server and work with all external resources (including terminal I/O operations, access to external data sources, interaction with other applications etc.) through universal language-independent Ubiq Mobile API. So, programmers can develop Ubiq Mobile applications in any language supported by .NET platform.

III. UBIQ MOBILE NETWORK

The architecture of Ubiq Mobile platform assumes that many different services can run on the same terminal server. The services are represented for mobile users in terms of logical channels that are conceptually close to TV channels – the user can switch them back and forth during his/her mobile terminal session. There may be several channels on the server; each channel is associated with its own type of online application. When the user switches channels, the application associated with the "old" channel would be suspended, while the application corresponding to the "new" channel would be activated.

The mobile client gets access to online services through a directory – a list of services available on the server. The server sends a copy of the directory to the mobile client during user's first login and checks version of the client copy during all subsequent logins. If client copy grows obsolete, the server sends newer version of directory to the client.

Several terminal servers can be combined into a network so that from users' prospective, they provide the united set of mobile services listed in the common directory. The concrete distribution of services among servers is transparent for the users – the user can just select from the common directory one of the services available and switch to the appropriate channel, even if such switching requires reconnection to another server.

The "primary" copy of common directory is storing in a special node of the network – a registration center. In case of either adding a new server to the network or updating a list of services running on existing servers, an appropriate request should be sent to a registration center. After performing necessary checks, the primary copy of directory would be updated and then replicated to all servers of the network. Furthermore, the new updates would be spreading onto mobile clients connected to the servers.

In addition to public services, Ubiq Mobile platform lets developers create private services with restricted access that require full-scale user authentication. All private services are listed in the separate directory that is stored in the registration center. Copies of this directory are distributed among servers and their clients in the same way as copies of the common directory.

IV. UBIQ MOBILE SERVER ARCHITECTURE

The server software includes the following main components:

- Communicator
- Application manager and user applications (services)
- Graphics API

- Mediators for access to external data and services

A. Communicator

Communicator is an internal server process that is responsible for working with all mobile connections on the server. It is launched during server startup and performs all necessary work on processing mobile IP connections and data transfer between the server and its mobile clients. The main functions of communicator are establishing new connections by mobile clients' requests, authentication of mobile clients, performing data exchange with connected clients and some other activities.

One of the important functions of the communicator is its participation in processing of unexpected mobile connection breaks. In Ubiq Mobile platform, we distinguish between two different situations – short-term disconnection, lasting from several seconds up to 3-5 minutes, and long-term disconnection, lasting longer. Short-term disconnections may happen, for instance, during user's movement in a subway. Long-term disconnection, as a rule, means real connection break and should be processed appropriately.

Handling of all short-term disconnections is localized inside the communicator that therefore is fully transparent for users' applications – from the application viewpoint, each short-term disconnection is just a delay in data exchange. As for long-term connection breaks, they are regarded as real disconnections and the corresponding user applications are suspended. After restoring connection and re-login of the disconnected user, the applications assigned to this user are resumed.

B. Application manager and user applications

Application manager is an internal server process that performs all the work related to management of user applications – starting new applications, suspending inactive applications and their further resuming, inter-application cooperation, termination of applications and some other functions. Like the communicator, application manager is started during server launch and works all the time until server stops.

User applications are .NET assemblies containing external classes that are attached to server logical channels and registered in the service directory. When the user first logs into Ubiq Mobile network and switches to some channel, the application manager loads appropriate class from the assembly (according to directory information) and creates new instance of this class for new user session.

Application manager can launch user applications either in synchronous or asynchronous mode. In synchronous mode, each application instance runs on its own thread and accesses the platform API functions via synchronized calls – after submitting any request for service through API function, the application thread execution is suspended awaiting results of this request. In asynchronous mode, applications are just class instances that all are running on application manager thread. The control flow of asynchronous applications is "state machine-like": it consists of set of instantaneous activities (which contain neither delays nor waiting for external events), and each of them ends with the call of dispatcher procedure that transfers execution flow to the application manager. When some external event occurs, the application manager calls a special method (event handler) of corresponding application for handling the event. The handler method is the same for all types of events; further branching is performed inside the method, depending on event type and, if needed, using state variables.

Synchronous applications are easier to program but creation of a separate thread for each application instance (corresponding to the user session) may lead to the overload of the server,

especially in case of really mass services, with thousands of mobile users. For potentially popular services with many users the asynchronous mode is much more suitable despite more difficulties and additional efforts in development and debugging of such services. The platform API calls are the same for both application modes – all the differences are encapsulated in internal methods and functions that are hidden from application developers.

For managing applications and controlling their requests' execution, the application manager provides its own scheduling mechanism that works both with synchronous and asynchronous applications in the same way. In addition to the standard set of application control features, the scheduler supports user session persistence – i.e., applications can be suspended for a long time (when the user either disconnects or switches to another channel) and then resumed, preserving full state of the user session. Session persistence is one of the basic mechanisms of Ubiq Mobile platform that is widely used.

C. Graphics API

Terminal I/O operations between applications and user mobile clients are performed through graphics API provided by Ubiq Mobile platform. Graphics API is not a separate process inside the server but rather a set of classes with methods available for user applications.

The whole terminal output is performed in a graphical mode. The application draws graphics onto a canvas – a graphical buffer located in server memory. The server automatically maintains the correspondence between the image on the canvas and on the screen of mobile device through the protocol. The size of the image on the canvas may differ from the screen size, so the screen of the mobile device is regarded as a "window" through which the user can see the image. The user can move the window along the canvas using scroll keys on the mobile device.

The canvas has multi-layered logical structure. Currently Ubiq Mobile platform supports five layers – a background ("wallpaper") layer, a layer of bitmapped images, a layer of text and line images, a layer of control elements and a layer of sprites. The image on the canvas is the result of applying layers to each other. Some layers have transparent background (like line images layer) while the others layers are opaque. All layers have the same logical size in pixels (equal to canvas size) and common coordinate system.

Different types of graphics data are drawn on different layers of canvas using different API methods. For instance, the application can draw a text string using either text layer (so, the whole font rendering will be executed on server) or controls layer – considering the output string as a text label. In the last case, the rendering will be executed on client side.

When the server performs terminal output, only those fragments of image, that will be visible on the screen of user's mobile device, are transferred (actually, a little bit more fragments for the purposes of buffering image on client side). For rapid "on-the-fly" selection of such fragments, special markup, depending on the client screen size, is imposed during canvas initialization.

For the purposes of traffic optimization, image fragments are compressed when being transferred via protocol. For different layers different compression methods are used, that allows to achieve maximum efficiency.

The terminal input consists of user events (like pressed functional keys or selected controls) and some internal service commands that are hidden from the applications. Service commands may contain requests for loading new fragments of the current image, for switching channels, for the information about current client state (for the purposes of user session persistence) etc. When some user event takes place, the corresponding application is activated and event handler method of the application is asynchronously called.

D. Mediators for access to external data and services

Besides terminal I/O, all the other interactions of user applications with the outside world are performed through mediators – service processes that constantly run on the server. Mediators provide applications with the access to the external data (like HTML and XML, external files, databases etc.) as well as to external services (TCP/IP, RSS feeds, Web services etc.). Each mediator works with its own type of data source serving all applications. The applications work with mediators via platform API.

The API for access to the external data sources is built on the basis of the unified data access pattern called DAMP. The DAMP pattern supports simple set of operations for processing abstract data – such as Open, Read and Write, Close, Seek etc. The set of basic operations is the same for all supported types of data sources. At the same time, DAMP pattern allows the user to customize external data, structuring them in his/her personal way. In such case all standard DAMP operations would work with structured data performing necessary filtering and marshalling actions during I/O operations.

Interface with external services is more diverse than the interface with the data sources, and therefore can not be formalized within a single pattern. For some types of services providing sort of data access (like RSS) we can use universal DAMP pattern whereas for more complex services it is impossible and the related mediators provide their own API.

V. CLIENT ARCHITECTURE

Ubiq Mobile clients are relationally simple graphical terminal programs that run on users' mobile devices and perform the following functions:

- Displaying on the screen graphical information received from the server through a protocol
- Support of scrolling functions – moving screen "window" along the canvas
- Sending information about user events (pressed functional keys, selected control elements etc.) to the server
- Work with common directory of available services; switching client session from channel to channel.
- Processing breaks of mobile connections

Currently Ubiq Mobile clients are implemented for Symbian S60, Windows Mobile and J2ME MIDP2 platforms.

The client program includes the following main components:

- Client engine (dispatcher)
- TCP/IP reader and writer
- Screen buffer
- Image decoder
- Special functions engine

The client is implemented as state machine. The client engine (dispatcher) is a central component that receives and processes all asynchronous events – pressed keys, incoming TCP/IP packages, timer events, etc. During event processing, the dispatcher changes state variables and invokes other components for substantial handling of the event. Mechanisms of components invocation may differ depending on particular components and target platform – either simple method call, or asynchronous inter-thread communication.

There are two separate components – TCP/IP reader and TCP/IP writer – that perform buffered I/O operations through TCP/IP. Both reader and writer work asynchronously with respect to dispatcher and can be implemented on separate threads. They communicate with dispatcher through mechanisms of inter-process cooperation.

Screen buffer is a structured memory area containing fragments of current graphical image that are displayed on the mobile device screen. Actually screen buffer size is slightly larger than physical screen size that allows client to make pre-buffering for the purposes of smooth scrolling of the image. The screen buffer has one-layer structure; the image storing in the buffer is a result of blending all canvas layers into a single bitmap. In addition to graphical information, the screen buffer contains special data structures describing control elements, sprites and components of background layer.

Fragments of images are coming from the server in compressed mode – so, the client should decompress them before blending into screen buffer. There is special component – image decoder – that performs decompression of all packed data received from the server. Image decoder works asynchronously and sends notifications to the dispatcher after decoding each fragment.

Special functions engine is a component that calculates special device-dependent functions (like GPS positioning). Since such calculations can take sufficient time (for example, because of waiting for external events), special functions engine is running asynchronously with respect to the client engine.

VI. Protocol structure

For data exchange between server and mobile clients, Ubiq Mobile platform uses custom binary protocol built over TCP/IP. The main reason for using proprietary custom protocol instead of one of standard open protocols was the necessity of maximal data transfer optimization – in particular, by implicit use of information about canvas logical structure during data exchange between server and its clients.

Data is transferring via protocol in the form of sequences of binary variable-length commands. Each command includes a header and a serialized tree of variable-length sections that contain semantically different data. Examples of sections are login-related data, image transfer-related data, event-related data etc.

Sections are represented in the command body as continuous data areas of variable length. Each section contains its length and the unique section code stored in the section header. So if the client doesn't support processing of some section inside the command, it can just skip it and switch to the next section of that command.

Sections can include subsections. The binary representation of subsections inside the command is similar to sections – no special tags or signatures for marking subsections are used. All necessary information for correct analysis of the command tree can be calculated on the basis of sections lengths.

The structure of data inside sections is not defined on the protocol level – it is assumed to be known both for server and client. The section usually contains some static fixed-length "head" and one or more variable-length "tails". Each variable-length portion of data contains its length as well as information about compression method used for packing this data. The complete list of compression methods and their unique codes is fixed on the platform level.

VII. TARGET APPLICATION TYPES

Terminal nature of the Ubiq Mobile platform determines its strengths and weaknesses. The platform provides mobile applications with proper degree of interactivity, rich user experience, access to device-dependent functions and ability to work everywhere – including slow networks and simple and relationally cheap mobile devices. The main weaknesses of the platform are its inability to work offline and mostly static character of the UI (coming from relationally long response time, even in fast networks, that prevents from any kind of dynamic screen animation). Advantages and restrictions of the Ubiq Mobile platform substantially determine the class of our "target" applications where the use of our platform will be the most appropriate. From our prospective, the "target" class includes the following categories of mobile services:

A. Information services with dynamic content

The main examples of such type of services are dynamic information displays of different kinds – arrivals and departures boards in the airports and railway stations, sports scoreboards, stock quotes displays etc. From one side, such boards are quite simple and straightforward in their logic and contain mostly text information. At the same time, the information is regularly updated and, if the mobile service is implemented as Web-based application, advanced technologies like Mobile Ajax are required to promptly display all updates on the screen of mobile device. The level of interactivity of the Ubiq Mobile platform, with its response time around one second, makes implementation of such kind of applications easy and efficient.

B. Complex mashups

Mashups (complex services that are results of superposition of several elementary services [4]) are getting more and more popular in the modern Internet. The main obstacle preventing such sort of services from moving to mobile devices is their high resource consumption – regarding both computational power and network bandwidth. In our platform, all calculations and getting information for creating mashup are performing on the server side while the mobile device just displays the resulting picture – without any additional overhead.

C. Mobile banking services

The main characteristics of the Ubiq Mobile platform – its efficiency, ability to work everywhere, high interactivity, proprietary custom protocol etc. – fully meet the basic requirements for mobile banking systems. The terminal architecture of the platform is an additional advantage in terms of development of mobile banking services because it allows to minimize the amount of essential information that is transferred between server and mobile clients. The protocol can be specially secured for use in banking services.

D. Location-based services

Ubiq Mobile platform provides positioning functions for those client devices that support this feature. This offers wide opportunities for creating location-base services. Location-dependent schedule of local commuter trains, location-dependent weather forecast and various services looking for the nearby-located facilities like shops, restaurants, ATMs etc. are typical examples of such class of services.

E. Multi-user game services

Ubiq Mobile platform provides an easy development of services that require intensive interaction between several users – for instance, multi-user games. High degree of interactivity and support of inter-application cooperation provided by application manager allow user applications to exchange information in real-time mode.

F. Mobile device as remote control

Rich graphics capabilities and high reactivity of the platform can be efficiently leveraged in mobile services where the mobile device acts as a real-time monitor and remote control operating on automated system like, for instance, country house automatics. The input data coming from the controlled system (like temperature inside the house) can be presented in nice-looking graphical mode. The service can run on the mobile device permanently (being most time in waiting mode) and signal to the user if some of the characteristics of the controlled system are going beyond the prescribed limits.

VIII. Conclusion

The Ubiq Mobile platform has been developed in the Department of Software Engineering of St.Petersburg State University as a research project. Currently there is a version of server software running under Microsoft .NET environment and client versions for the three popular mobile environments – Symbian S60, Windows Mobile and J2ME MIDP2. University students developed several test applications that we use for demonstration and debugging purposes. Our experience with current version of the platform confirmed correctness of our basic approach. The main working characteristics of the platform are in line with our initial expectations; test applications can properly work with different mobile networks, including slow GPRS connections.

Our nearest plans include organization of intensive beta-testing of the platform by the University students. For this purpose, we are developing set of Web 2.0 services [5] specially targeted to the student audience. After launching those services in the University, we expect to have at least several hundred users. At the same time, we are planning to grant free access to the clients' source code for beta testers and stimulate its porting and testing on different types of mobile phones.

REFERENCES

[1] Andrew S.Tanenbaum, Maarten Van Steen, "Distributed Systems: Principles and Paradigms", 2nd Edition, Prentice Hall, 2007.

[2] Brian Fling. "Mobile Desing and Development: Practical concepts and technoques for creation mobile sites and web apps", O'Reilly Media, 2009

[3] http://java.sun.com/javame/reference/docs/msoa.pdf

[4] J.Jeffrey Hanson, "Mashups: Strategies for the Modern Enterprise", Addison-Wesley Professional, 2009.

[5] James Governor, Dion Hinchcliffe, Duane Nickull, "Web 2.0 Architectures", O'Reilly Media, 2009.