High-level Componentization as a Way of Efficient Server-side Logic Implementation in Ubiq Mobile Platform

Alexandra Grazhevskaja
Saint-Petersburg National Research University of Information Technologies, Mechanics and Optics
Saint-Petersburg, Russia
s.grazhevskaja@ubiqmobile.com

Valentin Onossovski, Dmitriy Timokhin
Saint-Petersburg State University
Saint-Petersburg, Russia
v.onossovski@ubiqmobile.com,
dmitriy.timokhin@ubiqmobile.com

Abstract—**In this paper we consider approach to building server-side backend of distributed mobile applications on the basis of relatively big universal components. The proposed approach is implemented within Ubiq Mobile platform – a cross-platform technology for easy development of distributed mobile applications that can be treated as a proof of described approach concept. Several examples of "high-level integration" components and their particular implementation within Ubiq Mobile platform are considered in details.**

I. Introduction

The total "mobilization" of modern information systems, which is demonstrated in, on the one hand, in expansion of client's components from traditional computers to various mobile devices, and, on the other hand, in the migration of backend server logic from dedicated servers to clouds, is one of the major trends in the modern IT. While the applications and systems that are focused on personal use (social, entertainment, content delivery applications etc.) show tremendous progress in going mobile, the "mobilization" of business applications can boast significantly slighter success. The problem is that practically all applications for business are in fact distributed systems, combining rich client UI abilities with complex server-side business logic, and the development of such systems is much more complex task than creation of average mobile applications. This limits the usage possibility of popular tools for easy mobile development (such as iBuildApp [1] and other similar mobile apps constructors) because such technologies are fully concentrated on pure mobile development providing minimal integration with server-side backend logic (backend is considered in such applications mostly as a source of content). From the other side, various Web-based technologies that are used in conventional business programming for building distributed applications – such as Ajax – are not so efficient on mobile devices because of poor UI abilities and big traffic consumption.

The importance of adequate and easy-to-develop backend for mobile applications was realized by IT industry several years ago. Many companies appeared on the market since 2009 which provided libraries of backend components specially purposed for mobile applications usage. One of the most prominent companies was StackMob [2] that offered rich and universal library of backend components and provided cloud-based backend hosting for mobile applications developers (StackMob has been recently acquired by PayPal). FeedHenry [3] is another example of company that provides library of backend components. Recently, Google offered its own cloud backend platform for Android developers – App Engine [4] – that provides Android applications with wide set of backend functions such as authentication, messaging (including push notifications), different kinds of querying etc. Due to importance of this area, the new market niche – Mobile Backend As A Service (MBaaS) was separated from MEAP (Mobile Application Enterprise Platforms) sector. A brief review of MBaaS niche specifics can be found, for example, in [5]. A very detailed review of modern cross-platform tools and technologies for mobile development (that are close to MBaaS tools) is given in [6]. The majority of MBaaS consist of relatively young startup companies founded several years ago.

A common feature of the majority of MBaaS platforms is relatively low level of functions which they provide. The typical common set of functions includes push notification, data queries, file storage and sharing, messaging, authentication, access to social software, location services, etc. The complexity of business logic that can be built over such libraries is sufficiently limited – for example, it is not easy to implement various kinds of inter-user interactions within multi-user game on the basis of typical MBaaS platform. A low level of functionality, provided by MBaaS platforms, makes their specialization difficult in different business verticals (although workflows within each particular vertical typically have many elements in common) because the level of business process items is higher than the level of functionality provided by platform. Thus, the developers are forced to build high-level workflow items on client side due to insufficient level functions provided by MBaaS platforms.

II. "High-level integration" approach

In order to understand the limitations of existing approaches to the development of distributed mobile applications for business and, in particular, limitation of existing MBaaS tools to build backend for such applications (like StackMob, FeedHenry, App Engine), we will outline the main features which a modern mobile application should have, depending on its functional and non-functional requirements, complexity and objectives. Also, it will let us formulate the actual requirements for the most capable, effective and integral approach to the development of distributed mobile applications for business.

A. Modern mobile application

Modern mobile applications, developed for various platforms such as Apple iOS, Windows Phone, Android and other mobile platforms, can be divided into three groups, depending on complexity of their features.

1) Simple applications that are focused on personal use (social, entertainment, content delivery applications etc.) and include most of the following features, generally:

- Simple static UI screens that include standard UI controls.
- Simple business logic could be implemented in a client native manner by using standard platform-specific development tools and easily portable to other platforms.
- Workflow can be described within simple diagram in terms of transitions between static screens. Synchronous / asynchronous requests to server-side logic could be performed. Requests from the server can be naturally carried out by push-notifications.
- Single-user application. Simple interaction between users or application instances on different devices is possible, for example, through asynchronous shared access to data.
- Minor flows of simple data model and low requirements for bandwidth capabilities of the channel.

2) Average complexity applications that include small number of features listed below:

- Rich UI components, third-party UI libraries to be used.
- External system APIs and data feed providers connection and integration (maps, payment, social network, databases and data feeds, and other business systems).
- Complex business logic, which could be hardly implemented on client side.
- Complex and partially implicit workflow elements with asynchronous requests from the server, screen content generation.
- Complex dynamic interaction between two or more instances of an application on different devices (possibly on different platforms). A large number of concurrent users.
- Working with large volumes of complex data model, high requirements for mobile traffic channel bandwidth capabilities.

3) Distributed mobile applications for business include most features of average complexity applications and in addition:

- Full variety of platforms, devices and screen resolutions support.
- Application product line options and versions depending on business process particular use case, profiling and customization.
- High reliability and fault tolerance.

In order to implement the distributed business mobile application features described above we can use different approaches and technologies (with their own pros and cons) to create mobile application together with its server-side backend. Let's consider some of them in detail.

B. App constructors-based approach

The first way is to use one of the popular tools for easy mobile development, such as iBuildApp or other similar "mobile apps constructors", together with standard tools for specific mobile platform development. Many such tools are listed and described in details in "Cross-Platform developer tools" report by Vision Mobile [6].

1) Pros:

- Set of "ready to wear" UI preset components for basic feature list.
- Ability to implement simple business logic by describing "screen flow" transitions.
- Content-oriented usage of external data source.

2) Cons:

- No "out of the box" option for rich UI implementation.

- Difficult or even impossible to implement complex business logic that goes beyond simple screen flow transitions.
- Poor support for server-side and client-side business logic and workflow options, no external module or API access support.
- No custom multiuser support.
- Data model is used implicitly.

3) Conclusion: Mobile app constructors allow us to design and implement simple mobile applications from scratch and to deploy them to different mobile platforms. It could be done promptly by unqualified user who has no programming skills. So it well suits mobile development of applications from the first category, but in case of development of business applications it can be used for fast prototyping only.

C. Existing MBaaS tools-based approach

This mobile application development approach combines use of standard tools for client-side development and a set of server-side functions provided by MBaaS systems to construct server-side backend.

1) Pros:

- It is possible to use standard UI development tools and third-party libraries.
- MBaaS systems provide rather universal set of low-level server-side functions that can efficiently solve low-level business logic problems.

2) Cons:

- No special mobile UI integration provided.
- No support for access to external systems and APIs.
- No high-level server-side functions provided by MbaaS.
- Complex business logic to be implemented on the client-side, causing performance loss and unnatural application architecture implementation.
- It's hard to implement complex multiuser interaction.

3) Conclusion: The described approach suits well for developing applications of simple and average complexity, but can hardly be used efficiently when implementing distributed business mobile application, since it limits options for complex server-side business logic development, and doesn't provide level for business process modeling. Also it provides no rich UI support.

D. Enterprise level solution and web services-based approach

This approach is based on usage of enterprise-level technologies and solutions (Java, .Net, PHP etc.) as a data source (content generation) and as a business logic implementation host with remote access and web services provided as a connection protocol for mobile application developed using standard platform specific tools. It is typical approach when enterprise solution already exists and the problem is to make its services available via mobile application. Also this approach can be improved by using of so-called hybrid mobile application frameworks. The hybrid applications and appropriate development tools that are currently on the market are described in [6,7].

1) Pros:

- Evident benefit is easy integration to existent corporate solutions.
- Web technologies can be used for UI implementation and generation. Native-looking UI based on Web interfaces can be developed using hybrid mobile application frameworks.
- Enterprise solutions are specially purposed for complex business logic development, including external systems access and business functions modeling on any level of abstraction.

2) Cons:

- Complex workflow development options are limited by communication protocol that is based on web services. The use of web services considerably increases traffic consumption.
- Server-side event management, implicit workflow and server-driven screen content generation can hardly be handled in mobile application.

3) Conclusion: Generally, the approach seems to be suitable for development of complex distributed mobile applications, but the resulting applications turn out to be too "heavy". Another problem is caused by inefficient and poor Web UI that forces developers to use additional "hybrid" frameworks for making UI more native-looking.

E. "High level integration" approach

To overcome the limitations of the approaches described above and in order to succeed from their benefits we propose a new approach called "high level integration" approach. It combines the opportunities provided by enterprise solutions and technologies with flexibility and usability of existing MBaaS tools. The fundamental idea of the approach is to develop core business logic comprised of a number of relatively big server-side components, which are highly integrated, provide complete set of functions of different levels of abstraction and are customizable enough for effective development of a wide range of mobile applications.

This approach allows us to design complex business logic, providing backend of any level of abstraction and integration complexity. Also, we can use set of server-side logic components, which can be easily accessed from application with no unnecessary traffic consumption and without workflow limitations coming from server components. This set of server-side logic components is rather universal and customizable for particular business process to be implemented. Benefits of "mobile app constructors" - prompt abilities of prototyping, development and deployment - can be taken easily in our approach by implementation of the following features: addition of the server-side and client-side customizable components in one click, deployment to different mobile platforms and UI modeling tool.

High level integration approach can be implemented within various frameworks. It implies the following basic requirements to the "host" environment:

- Platform-level support of relatively big server-side components that can run in server environment asynchronously and independently (like Java Beans, from example).
- Simple and efficient mechanism of interaction between these components that can be used for building distributed applications (like messaging, remote calls or some other).
- It is very desirable to have an extendable IDE for providing support of integrated components on the IDE level (within specialized add-on).

We try to implement high level integration approach within Ubiq Mobile platform – a cross-platform technology for easy development of distributed mobile applications [5], [6]. This implementation can be treated as a proof of concept for the described approach. We believe that high level integration approach can significantly simplify distributed business mobile application development, as if it was done using app constructor but had a power of complex distributed application.

III. Componentization in Ubiq Mobile platform

Ubiq Mobile is a universal platform for easy creation of distributed mobile online services and applications [8,9]. The platform encapsulates most of the technical details that are specific for different mobile devices. It provides simple and intuitive API for the distributed mobile online applications and services development. One of the key ideas of Ubiq Mobile platform is the usage of ultra-thin client-based "mainframe-like" architecture, where the mobile devices are used like graphic terminals, and the majority of applications' business logic is implemented on the server.

Ubiq Mobile is a cross-platform system from the viewpoint of supported types of mobile devices. Ubiq Mobile applications can work on various mobile platforms, including iOS, Android, Windows Phone and JavaME (particularly, on Nokia Asha phones). The server part of the system is based on Microsoft technologies (server-side components are running under .NET in Microsoft Azure cloud) and, thus, can be considered as vendor-locked one. But from the prospective of this paper it is not significant, because the basic approach, proposed in the paper, can be implemented within any platform that satisfies basic requirements to the "host" environment listed above. So, the proposed approach to backend componentization is absolutely a cross-platform approach in its essence.

Ubiq Mobile architecture with ultra-thin client determines basic strengths and weaknesses of the system. On the one hand, concentration of business logic on the server side simplifies development of complex distributed applications with mobile access. At the same time, it significantly extends variety of supported mobile devices, including very simple ones. On the other hand, terminal-like architecture causes lower reactivity and bigger response timein comparison with "traditional" client-centric mobile apps. These pros and contras determine the category of applications which Ubiq Mobile is targeted to. It includes complex distributed applications with sophisticated business logic, purposed mostly for business use, that should work on various mobile devices and don't require very quick "real-time" response and animated UI. Interfaces to business systems, multi-user mobile games and social applications are examples of apps and services which belong to this category.

The common problem for all systems with thin client-based architecture is their poor ability to work offline. In Ubiq Mobile, some measures have been taken to mitigate the inconvenience caused by this problem: saving full state of users' mobile sessions during connection breaks; caching images and other unchanged data on client side, and some others. But unfortunately, such measures cannot provide smooth work of non-trivial fragments of business logic in the absence of connection with the server.

Ubiq Mobile is aimed at solving the same sort of problems as WidSets technology that was developed by Nokia several years ago [10]. But from architectural point of view, the two technologies use rather different approaches: the basic idea of WidSets is free roaming of lightweight device-independent program components ("widgets") between mobile devices of different types. So, WidSets applications are rather "client-centric". On the contrary, the basic idea of Ubiq Mobile is maximum concentration of business logic on server side and set of unified ultra-thin clients on mobile devices of different types.

There are two types of applications in Ubiq Mobile system – custom applications and application services. Custom applications are related to mobile users' sessions.

The application services are started during the server start and remain running during the whole lifetime of the system. The Ubiq Mobile applications are separately executed components (specifically - .NET assemblies) that are dynamically deployed onto the server and running in its environment. Each application runs in a separate thread using a thread pool.

The Ubiq Mobile server core implements basic functionality of the system. It ensures communication with clients' mobile devices, deployment of applications and services, their running within server environment and their mutual interaction.

The Ubiq Mobile platform provides a set of system services over server core – such as user authentication, persistence support, geocoding and some other services. Every such service is available for application developers through its API. Similar APIs are used for access to the external systems and data sources.

The interaction between custom applications and the platform application services is messaging-based – both synchronous and asynchronous messages are available.

The mobile ultra-thin clients interact with the server via proprietary binary protocol built over TCP/IP. The protocol is specially optimized for mobile connections; it includes special mechanisms for handling possible mobile connections breaks and mobile traffic reduction.

Development tools for Ubiq Mobile platform are "packed" into a special plug-in for Microsoft Visual Studio. The plug-in includes visual DSL editor for creating component architecture of the developed application, visual UI designer for cross-platform UI development, universal mobile client emulator integrated with Visual Studio debugger, tools for deployment of the developed applications into a cloud and some other tools. Ubiq Mobile plug-in provides comfortable environment for efficient development of complex distributed applications with access from various mobile devices.

Since the abilities of Ubiq Mobile platform completely satisfy to the basic requirements of high-level componentization, it was decided to use this approach for building complex Ubiq Mobile applications. Integrated components are implemented in Ubiq Mobile as service applications, publishing their APIs to other applications for interaction with them. Inside APIs, messaging is used as a low-level mechanism for inter-component communication.

High-level componentization is supported in Ubiq Mobile platform on the level of development environment – set of components are contained in the standard plug-in component library, and can be easily included into architecture diagrams of the developed applications. Interfaces between these integrated components and other components of the developed application, are automatically generated by the plug-in.

We started from developing "general-purpose" integrated components – Dispatcher and Authentication services. Dispatcher component provides basic functionality for inter-user communication within multi-user Ubiq Mobile applications while Authentication component provides various types of authentication for mobile users. The results of developing both integrated components and their use in real application can be evaluated as quite successful, and we are currently planning to extend the set of integrated components provided for Ubiq Mobile developers.

IV. Dispatcher component

As an example let's consider Dispatcher Service component, that encapsulates basic tasks associated with the interaction between mobile users in the distributed multi-user applications.

Such tasks as user registration, user authentication, user interaction, storing information about users and their interactions in some sort of persistent database, etc. are common for any multi-user mobile application and do not depend on its business logic features. Thus, the componentization of such functionality (rather complicated to implement and debug) exempts developers from significant part of routine work.

The logical model of user interactions through Dispatcher includes two classes of objects: users and dialogs. User objects represent mobile users and dialog objects represent established connections between them, which allow them to exchange with messages. Both one-to-one and many-to-many dialogs are supported. Users can also subscribe to each other to be notified about peers' updates. Any changes in peer status as well as application-dependent information changes cause such notifications.

User object contains the following attributes:

- User name.
- UserID.
- Password.
- User status.
- Object with user-specific additional information (for instance, for multi-users game – information about player's score).

Dialog object contains the following attributes:

- DialogID.
- Dialog status.
- Object with additional information (for instance, message history of the dialog).

Fig. 1 shows Dispatcher Service and user applications interaction.

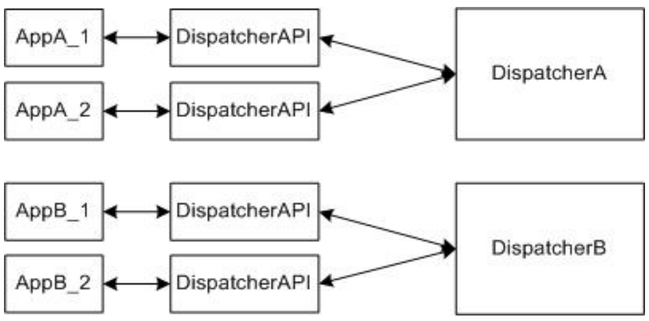

Fig. 1. Dispatcher and user application interaction

Dispatcher is an application service running in separate thread. There may be several Dispatcher instances for different types of user applications. User applications interact with Dispatcher via locally instantiated DispatcherAPI objects. Each call of DispatcherAPI method cause sending of the appropriate message to a Dispatcher. As a result of the method invocation, some meaningful data or an error code can be returned. Asynchronous messages coming from Dispatcher raise appropriate asynchronous events in API that can be handled by the application.

The DispatcherAPI functionality includes the following groups of functions:

- User login and registration methods.
- Methods for obtaining information about users.
- Dialogs processing methods.
- Inter-user communication methods.

Let’s consider how these methods work. We assume that some multi-user application needs to register a new user. For this purposes, it invokes Register method of Dispatcher API with appropriate parameters – user name and password and, probably, some object containing application-specific information (like game status of this user, for example). The Register method creates dispatcher message containing all passed parameters, and sent it synchronously to Dispatcher (i.e., applications thread is waiting until receiving response from Dispatcher). When Dispatcher receives the message from its input queue, it invokes appropriate message handler, depending on message type. All message handlers perform some initial validation of the parameters of incoming messages. For example, the user name obtained from Register message, is checked for uniqueness and the password is checked for compliance to supported authentication policy. For application-dependent additional data, validation is not performed because Dispatcher knows nothing about their semantics. Such additional data are just added to the database associated with the given user. After completion of registration process (either successful or unsuccessful), the handler sends back a response message to the application that requested Dispatcher for registration. In case of successful registration, the application receives a unique UserID that will be used for further communications with Dispatcher.

If the user is already registered in the system and wants to log in, the user application invokes one of two overloaded Login methods of Dispatcher API for authentication either by user name and password or by unique identifier of client’s mobile device. The Login request is handled by dispatcher in the same way like Register request – an appropriate handler is invoked, validation of parameters is performed and, in case of successful authentication, Dispatcher updates current status of the user from “Offline” to “Online” and sends back to the application a response message with unique UserID. After successful login, the user can update his application-dependent information by invoking UpdateUserInfo method with a single parameter - an object containing new application-specific information. This new value will be stored in the Dispatcher database. As a result of execution of this request, Dispatcher sends back to the application the result of database update operation(successful or unsuccessful) and, in addition, sends notification messages to all other users who are connected with the given user through dialogs. Such standard attributes as username, user ID, client mobile device ID and some others, are stored in DispatcherAPI object during its whole lifetime. From application developer prospective they can be considered as implicit parameters that are passed to the invoking API methods. When necessary, these attributes are copied into messages sending from API to Dispatcher. This mechanism reduces number of parameters passed to API methods and makes the whole API more compact and easy-to-use.

The user application can obtain list of current users (applying various filters – all, active, online etc.) by GetUserList method invocation. Type of applied filter is passed to the method as a parameter. The resulting list sending back by Dispatcher to the application contains basic information for each user – username, userID and current status of the user. It is possible to retrieve more detailed information about particular user through GetUserInfo method (one of overloaded variants).

User applications can communicate via Dispatcher by using dialogs. One of participants (communicating applications) creates the dialog by CreateDialog method invocation and then invites other participants to join this dialog via InviteUser method. The dialog between two users may be established by simpler way: the caller application just invokes InviteUser with two parameters UserID of callie application and application-dependent info. In this case, Dispatcher creates a new dialog, assigns unique DialogID and sends invitation message to the callie application.

After receiving invitation, the invited application may either accept or reject this invitation. Invitation is accepted by JoinDialog method invocation and rejected – by RejectInvitation method.

Applications can send any information messages within established dialog by SendMessage method invocation with two parameters – DialogID and user-defined object (containing sending information itself). Each message sent by one participant is delivered by Dispatcher to all dialog participants. Type of information that is sent via message depends on nature of application: for example, in multi-user game messages may contain players' turns, while for chat application it may be either text or file message.

We can see that Dispatcher efficiently encapsulates most of routine inter-user communication work for various classes of multi-user distributed applications. Moreover, it can be used for setting up communications between user applications and other applications, representing some external devices. For instance, a service of translation of images from remote webcams to mobile users' devices, can be implemented on the basis of Dispatcher.

Another typical example of using Dispatcher component is a generic template for turn-based multi-player games, such as Battle Ship, TicTacToe etc. The template is implemented as a state machine that performs such generic functions as user registration/logging in, displaying list of available players, invitation player to the game, making game turns, determining of winner, deciding whether play again or not, etc. All game-specific functionality (like game file initiation, making turns, determining winner) is represented by empty templates that should be overridden in particular game implementations - ancestors of the template. But most of general functionality is related to inter-user communications and the appropriate template functions are actively use DispatcherAPI. The architecture of multi-user game application built over game template, looks like shown on fig. 2.

We can see that in this application, DispatcherService is interacting with two other integrated components – DBServices (that provides persistent storage for applications) and AuthenticationService, providing various mechanisms for authentication (via local users database, OpenID, social networks etc.). These components are built on the same principles as Dispatcher component and provide their own APIs for access to them from applications. Like in DispatcherAPI, applications communicate with these components via messaging.

The functionality of the Dispatcher component can be extended in various directions. Currently we are working on Dispatcher extension called GeoDispatcher – the integrated component that handles inter-user communication based on their locations. In addition to basic Dispatcher functionality, the new component provides ability to get list of users sorted by their distances from the current user (of getting list of all available users located within some given range), users' locations tracking abilities and some other location-based functionality that helps significantly in building modern distributed mobile applications.

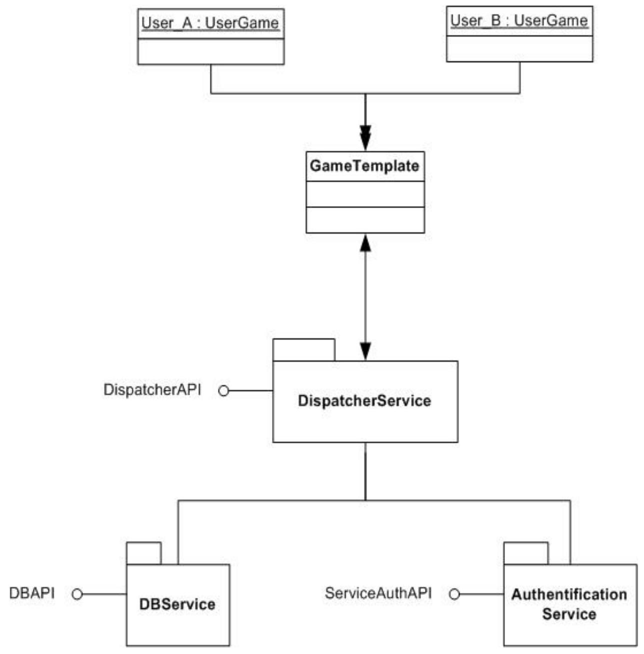

Fig. 2. Architecture of multi-user game application

V. Conclusion

In this paper we tried to consider the problem of building server-side backend for distributed mobile applications on the basis of high-level integrated components that encapsulate big self-sufficient fragments of application business logic. The proposed approach provides higher degree of integration of the components – "building blocks" for server-side backend in comparison with the majority of MBaaS systems. It allows to use highly integrated components for implementation of separated elements of business processes that is the basic goal of the proposed approach.

The particular implementation of the approach that has been made within Ubiq Mobile platform confirmed its effectiveness. One can imagine many other useful and efficient integrated components that cover typical needs of distributed mobile applications. Among possible examples of such components – support of electronic queues with push notifications to mobile users' devices, filters over external data sources/feeds that "monitor" current data streams and notify user when the data that meets some predefined conditions, is appeared, modules for monitoring locations for moving objects etc.

The proposed approach is not Ubiq Mobile-locked. It can be implemented over any distributed “host” environment that provides appropriate feature set and meets the requirements of host environment listed in Chapter II.

The use of the proposed approach lets mobile developers significantly facilitate the development of complex backend for their applications and provides opportunity to create libraries of components – fragments of standard business processes. The overall design of distributed application, using integrated components, can be easily implemented within IDE by using customizable graphical design tools.

There are two basic restrictions of the proposed approach – first, the host environment should be “advanced” and server-centric enough for implementing complex server-side business logic. Second, it inevitably requires from the developer to write a code – it is hard to imagine how to use integrated components together with simple mobile app constructor such as iBuildApp. However, this applies to the most of MBaaS systems.

The above example of implementation of the particular component – Dispatcher – gives an idea of the scale and level of complexity of the components that can be implemented with the proposed approach. In business applications, the proposed approach allows to build complex integrated components, which correspond to separate elements of the standard business processes that makes it possible to form component libraries specifically for different business verticals.

REFERENCES

[1] iBuildApp, Inc. official website, Web: http://ibuildapp.com.

[2] StackMob, Inc. official website, Web: http://www.stackmob.com.

[3] FeedHenry, Ltd. official website, Web: http://www.feedhenry.com. Google, Inc. official website, Web: https://developers.google.com/appengine.

[4] Andrew Brouwnberg. MBaaS Can Accelerate Mobile App Rollout – Analyst brief, Web: https://www.nsslabs.com/reports/mbaas-can-accelerate-mobile-app-rollout

[5] Cross-platform developer tools-2012. Report by Vision Mobile.com.Web: http://www.visionmobile.com/product/cross-platform-developer-tools-2012/

[6] Tata Consultancy Services. Hybrid Mobile Applications Development Approached. White paper. Web: http://www.tcs.com/resources/white_papers/Pages/Hybrid_mobile_application_development_approaches.aspx

[7] V. Onossovski, A.Terekhov, “Ubiq Mobile – a New Universal Platform for Mobile Online Services”, Proceedings of 6th seminar of FRUCT Program, 2009.

[8] T.Bryksin, Y.Linvinov, V.Onossovski, A.Terekhov, “Ubiq Mobile + QReal – a technology for development of distributed mobile services”, Proceedings of 10th conference of FRUCT association, 2011.

[9] Nokia Widsets technology. Developer Wiki. Web: http://developer.nokia.com/community/wiki/Category:WidSets