Cloud based Cross Platform Mobile Applications

Building and integrating cloud services with mobile client applications

Cosmina Ivan[1], Razvan Popa[2]

[1] **Department of computer science**
Technical University of Cluj Napoca
Cluj, Romania
cosmina.ivan@cs.utcluj.ro

[2] **Department of computer science**
Technical University of Cluj Napoca
Cluj, Romania
Ryan.Popa@yahoo.com

Abstract

Building mobile applications presents new challenges. There are various services and frameworks available to address some issues such as device fragmentation and computing power. Hence it is important to understand what tools developers have and how they can be integrated for productively creating high quality apps. Cross platform mobile development using Titanium allows faster time to market. The low computing power of mobile devices can be supplemented with the Cloud. Data synchronization in a distributed system requires a signaling mechanism such as SignalR. Third party services can provide plug in functionalities that can leverage the functionalities and improve user experience. Building a successful mobile application requires understanding each component of this mobile ecosystem and also how they can be linked together.
Keywords: *mobile; cloud; mobile analytics; mobile services; mobile integration*

1. Introduction

Worldwide smartphone adoption is increasing fast. Worldwide sales of new smartphone in Q1 of 2012 reached 149 million units compared to 101 million in Q1 if 2011, comparing to the fact that the total number of mobile phones (smartphone and regular phone) sold worldwide in Q1 of 2012 was 398 million units.

While there are over 1 million applications available today for iOS and Android combined, it is important to strive for best possible quality in the shortest amount of time. In order to create successful applications one must understand the infrastructure that stands behind complex applications, and what practices must be employed for productively creating a state of the art application.

While there is a lot of documentation on building applications, very few have concerned about the building components and the interaction between them. Also a successful developer needs to know what tools he has at his disposal for reducing the amount of work by using third party services or frameworks.

In this paper we are analyzing the most important components constituting the infrastructure and the present the tools we are suggesting for a productive development environment. We are also presenting how these components can interact with each other in a reliable and scalable way. As an example we will be using a taxi ordering application for smartphones, which will illustrate the concepts we are presenting in this paper.

The components of the mobile infrastructure together with the tools we are analyzing in this research paper include the client mobile component (build using the *Appcelerator Titanium* platform), proprietary cloud services exposed to the client (built on top of Windows Azure with WCF services), 3rd party services for analytics and push notifications, client signaling mechanism (*SignalR* for *Asp.Net*). We are also presenting what advantages and disadvantages these tools have and why we consider them as being productivity enhancers.

2. General architecture

The architecture for the system, which we are using as an example, is presented in Figure 1. There can be identified two types of components.

The first type includes the proprietary components. This need to be developed by the developers, are based on specific frameworks and include: the components inside the *Proprietary Cloud* (designating the service built by the developers which is hosted in the *Windows Azure Cloud* in our example), the mobile application, the browser client and the management interface [1].

The second type of components are represented by the *3rd party services* (distinctly from proprietary services, *3rd party services* or *external services* are built by other organizations and are made available to the public for use) are exposed trough certain *API*. In this example we are using 5 types of external services: The Notification Service – used for sending *Push Notifications* (a way of alerting a mobile device), social services that offer integration with *Facebook* and *Twitter*, *Analytics* services for tracking usage patterns, *SMS* gateway service for sending the user an *SMS* with the activation code (required by the business rules), and a geocoding and reverse geocoding service for converting *GPS* position into street address. All of these components will be explained together with their implementation technologies and internal working.

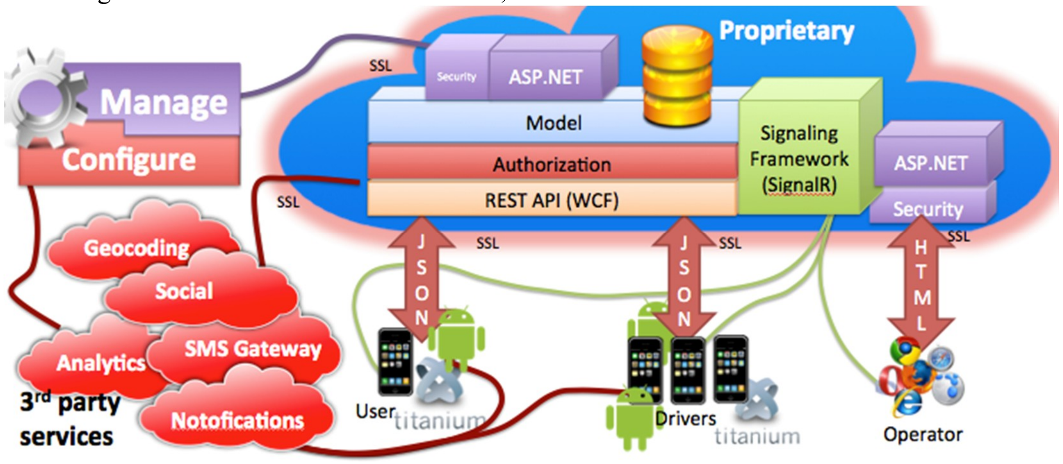


Fig 1: General Architecture of the Taxi Ordering Application

For understanding how the whole system works together, let's examine the most important functionality of our Order Taxi application: sending an order by the client.
When a user orders a taxi from it's mobile phone, a request is serialized as *JSON*, signed using the user key and sent to the server trough the server *API*.[8] The server determines the nearest driver to the user, by interrogating the database based on the business logic. Once the nearest driver is determined, the *Signaling Framework* (used to alert the mobile application) is used to determine the selected driver's app to download or update the order sent by the client. After the driver confirms the order, the server uses the same approach to inform the user that his order was responded. All requests sent to the proprietary server components need to use SSL for proving security to the system.

2.1.Native cross platform mobile development

In Q1 of 2012, Android had 50.6% market share, while iOS had 23.8% of new sales. This means that their combined market share is approximately ¾ of all new smartphone sales.
But even if the two players dominate the market, developing for these two frameworks is time consuming as very little work can be reused, the programming language and development environment for *Android* and *iOS* being very different. Also if another *OS* must be targeted in the future, this would require rewriting the application.

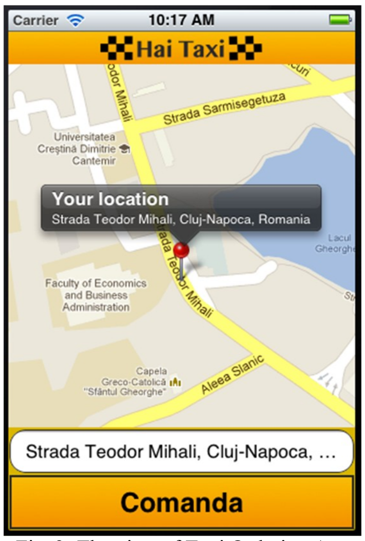
In order to increase productivity a cross platform approach can be used. There is currently a wide range of cross platform solutions available including *Appcelerator Titanium*, *RhoMobile* and *PhoneGap*. *Appcelerator Titanium* is an open source project and has integrated most common functionalities of *iOS* and *Android*. Developing with Titanium compared to other *Cross Platform* solutions is that it allows for native development. For example when using *PhoneGap* you are actually developing a web app that run in a WebView UI component and has access to active services provided by the platform. When using *Titanium* you are programming in *JavaScript* and the framework translates the code into native *Java* or *Objectual-C*.
Titanium currently supports two operating systems: Android and iOS. It also plans on fully supporting *BlackBerry*, which is currently in Beta. One disadvantage is the lack of support for *Windows Phone* at the moment, if this OS needs to be targeted.

The development environment for Titanium is using its own *IDE* based on *Eclipse*, *Titanium Studio*), and it uses the JavaScript development language. *Titanium* can also be extended using module in order to offer complete native functionality, which is not supported by the Titanium platform.

Using *JavaScript* as a development language offers the benefit of low entry level compared to *Objectual-C or Java*.
One difference when developing with *JavaScript* is that it requires understanding code organization when tackling large

projects in the context of a language that was previously used for writing small code blocks in browsers. The recommended way of development in Titanium is using a *MVC* pattern. Also applying *OOP* best practices as encapsulation is the recommended approach even weather JavaScript is or isn't object oriented is highly debated. Leveraging the *MVC* pattern will help produce cleaner code and promote reusability. Applying the *MVC* pattern within *Titanium* is relatively easy.
Considering the Order Taxi App, we will decompose it into its three components: Model, View, and Controller.

The *view* is responsible for creating the window and adding the necessary UI components. Views may optionally specify styling properties (color, font, and positioning). The recommended practice is to extract all styling properties into external *JSS* files, which promotes easy style modifications and customization. Figure 2 displays the view for the taxi ordering application. It is composed of a Window, on which are added a map, a textbox and a button. When the button is pressed it fires an event, which is handled by the controller.


Carrier
10:17 AM
Hai Taxi
Your location
Strada Teodor Mihali, Cluj-Napoca, Romania
Strada Sarmisegetuza
Universitatea Creştină Dimitrie Cantemir
Faculty of Economics and Business Administration
Capela Greco-Catolică "Sfântul Gheorghe"
Aleea Slanic
Strada Teodor Mihali, Cluj-Napoca, ...
Comanda
Fig. 2: The view of Taxi Ordering App

The *controller* is responsible for managing events (button taps) and navigation. For example when the tap event is registered for the Order button, a notification is sent o the model together with the data the user inputted in the textbox. Some developers use event handlers inside the controller to communicate with the server. While this practice is convenient for simple views, it leads to unorganized code when there are multiple services to be accessed. In such a scenario it is advisable to move the server interaction on the model where it can be encapsulated and offer better reusability.
The *model* is typically a JSON payload retrieved from a Restful service. For example, the *handleResponseEvent* will return response information from the server.
The advantages of using *MVC* in *Titanium* include better code organization and readability, promotes reusability and maintainability.

2.2.Signaling to client

While sending data is fairly easy, listening for incoming information requires new approaches given that users might be charged additionally by the carrier and to avoid increased battery drainage[1].
Traditionally web applications used a polling technique for getting updates. For example a stock ticker knows that there are updates every 10 seconds, so it makes sense to implement a timer that gets the updates every 10 seconds from the server.
For the Order Taxi application this is not feasible as the polling interval is unknown. For example the driver could wait for hours without receiving an order, so he would poll thousands of times to receive one favorable result. Also the polling interval would have to be below 2 seconds (to offer a response time as low as possible), which would mean a high demand on the server and low scalability.

SiganlR is an Asynchronous library for .NET to help build real-time, multi-user interactive applications and is based on long polling techniques.
Long polling is a variation of the traditional polling technique and allows emulation of an information push from a server to a client. With long polling, the client requests information from the server in a similar way to a normal poll. However, if the server does not have any information available for the client, instead of sending an empty response, the server holds the request and waits for some information to be available. Figure 3 illustrates this functionality by comparing time based polling and long pooling. Once the information becomes available (or after a suitable timeout), a complete response is sent to the client. The client will immediately re-request information from the server, so that the server will always have an available waiting request that it can use to deliver data in response to an event.

Long polling is itself not a push code, but can be used under circumstances where a real push is not possible.

Using long polling in the client mobile application gets notified to update it's data, and thus to synchronize with the server, exactly as the new data is available. Using a long polling approach there is a very low demand on the server which translates into a high scalability, as new users and drivers are added to the distributed taxi ordering system.

In the .NET framework SignalR represents an implementation of long polling. However SignalR does not rely only on the long polling technique. SignalR has a concept of transports, each transport decides how data is sent/recieved and how it connects and disconnects.

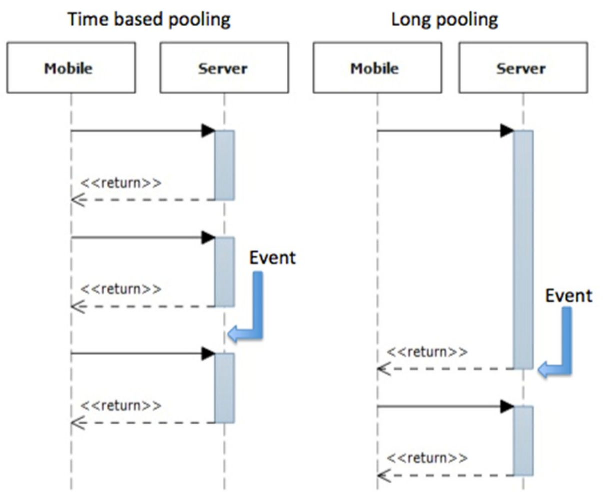


Fig. 3: Long Pooling vs. Time-based polling

Transports build into SignalR are: WebSockets, Server Sent Events, Forever Frame, Long polling. SignalR tries to choose the "best" connection supported by server and client (a desired connection can also be specified implicitly). However WebSockets and Server Sent Events are still not widely supported at present. [1]

The implementation of SignalR in Titanium, in Order Taxi app is by using a WebView, which triggers events once they arrive from the server. *SignalR* requires *jQuery* on the client side. On the server side it uses two concepts: *Hubs* and *persistent connections*. Taxi Ordering App uses *Hubs*. Extending the *Hub* class does implementation of a hub on the server. Each method defined inside this class is mapped to a *JavaScript* implementation at compile time. The *JavaScript* document is saved on the project root folder and the client must reference it. It is possible to have broadcast events, which alert all the clients or only one in particular. It is also possible to group client and inform a whole group at a time.

2.3.Push Notifications

Mobile apps are allowed to perform only very specific activities in the background, so battery life is conserved.
But there is required to be a way to alert the user of interesting things that are happening even if the user is not currently inside the app.

For example, maybe the user received a new tweet or their favorite team won the game. Since the app isn't currently running, it cannot check for these events. *Mobile OS* have provided a solution to this. Instead of the app continuously checking for events or doing work in the background, a server-side component for doing this can be written.

When an event of interest occurs, the server-side component can send the app a push notification, which is intended for signaling the app that there are event pending on the server. *Push Notifications* are not intended for transmitting data to the client app, but are used as a signaling mechanism when the app is not running. *Push Notifications* for *Android* are called *C2DM* (*Cloud* to device messaging).

Push Notifications makes no guarantee about delivery or the order of messages. So, for example, while you might use this feature to alert an instant messaging application that the user has new messages, you probably would not use it to pass the actual messages.

Figure 4 represents how push notifications work. Apps need to register with the *Notification Service* that is *APNS* (Apple *Push Notifications Service*) for Apple and *C2DN* for Android. This assigns a specific id called device token that the app can send to the server. If the server needs to perform a push notification, it uses this device token to specify the app it want to send the notification to. Device tokens change so it is advisable to perform device token updates on the mobile prefeably every time the app is opened. Users can also opt out of push notifications for the app or delete the app. In this scenario, push notification wil not reach the user. The Notification service offers an *API* that the server can poll peiodically to determine what device tokens are still active. Because communication between the server and the notification Service is not trivial, 3rd party services are available such as *Urban Airship*. Other operating systems also provide push notification, such as *Windows Phone* or B*lackberry*, the mechanism being the same.

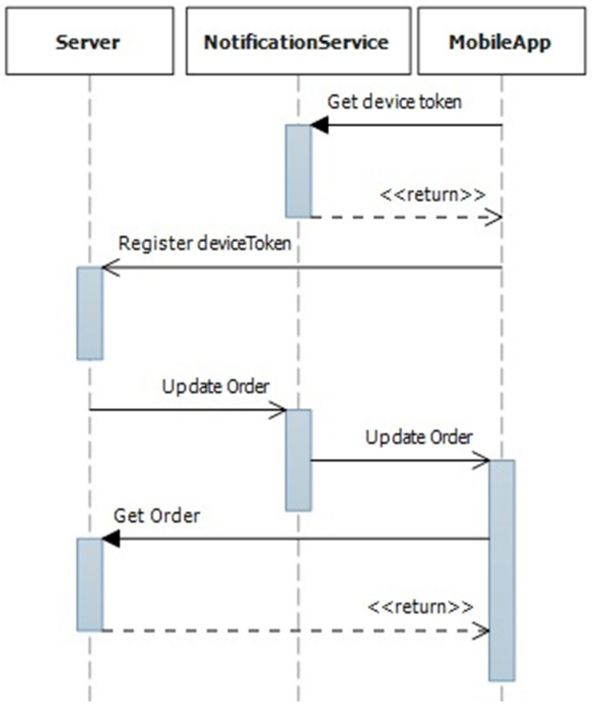


Figure 4: Notifications mechanism

3. Integration

3.1.Integration of the client with the server

Not all mobile applications are required to have a server component. One example is an application that is used to show the battery drain level. Though most applications have a service backend. If the service backend logic is fairly simple, for example if it is used solely for storing data, the developer can opt in for a 3rd party backend server such as *Parse.com* if the business logic is simple enough to allow this.
However applications that require a higher degree of complexity need to create their own model and an API for exposing the services. [2]

Mobile Cloud Computing (MCC) refers to an infrastructure where both the data storage and the data processing happen outside of the mobile device. Mobile cloud applications move the computing power and data storage away from mobile phones and into the cloud, bringing applications and mobile computing to not just smartphone users but also a much broader range of mobile subscribers [7].

Cloud computing is known to be a promising solution for mobile computing due to reasons including mobility, communication, and portability, reliability, security. In the following, we describe how the cloud can be used to overcome obstacles in mobile computing, thereby pointing out advantages of *MCC*.[3]

3.1.1.Extending battery lifetime

Battery is one of the main concerns for mobile devices. Several solutions have been proposed to enhance the CPU performance, and to manage the disk and screen in an intelligent manner [5] to reduce power consumption. However, these solutions require changes in the structure of mobile devices, or they require a new hardware that results in an increase of cost and may not be feasible for all mobile devices. Computation offloading technique is proposed with the objective to migrate the large computations and complex processing from resource-limited devices (i.e., mobile devices) to resourceful machines (i.e., servers in clouds). This avoids taking a long application execution time on mobile devices, which results in large amount of power consumption.

Studies evaluating large-scale numerical computations [5] shown that up to 45% of energy consumption can be reduced for large matrix calculation. As a consequence computationally intensive tasks should be delegated to the Cloud as CPU intensive work can drain the battery level significantly.

3.1.2.Improving data storage capacity

Storage capacity is also a constraint for mobile devices. MCC is developed to enable mobile users to store and access large data in the cloud. Examples of cloud services for doing this are: Azure Storage and Amazon S3. Storing data in the cloud also allows for data synchronization along multiple devices. iCloud is a service that allows work that is began on an iPad to be continued on a Mac for example. Also large database services are available which offer better scalability. These databases include rational databases (SQL Azure), highly scalable Table Storage databases and NoSql databases such as MongoDB.

3.1.3.Improving reliability

Storing data or running applications on clouds improves reliability since they are backed up on a number of computers. This reduces the chance of data and application lost on the mobile devices. CDN can be used to provision data as close to the customer as possible to improve access time. In addition, MCC can be designed as a comprehensive data security model for both service providers and users. The cloud can be used to protect copyrighted digital from abuse and unauthorized distribution [6] Also, the cloud can remotely provide mobile users with security services such as [9] virus scanning, malicious code detection, and authentication . Also, such cloud-based security services can make efficient use of the collected record from different users to improve the effectiveness of the services.

When considering mobile applications, *REST* services are preferred over *SOAP* as they are lighter and easier to implement. Among the REST services, there are two common types: *JSON* and *XML* based. *JSON* is especially preferred when developed with Titanium, as *JSON* objects are pure *JavaScript* objects. Also *JSON* representations are less verbose. *JSON REST APIs* are offered by services like Google, Facebook, Twitter, Flickr etc.

The Order Taxi application exposes *JSON REST* services using *WCF* Framework. Working with *JSON* in the context of *WCF* is easy as the framework automatically serializes *JSON* request and responses and performs data binding between *JavaScript* and *CLR* objects.

Once the data is received on the server, the server performs some security check on the request. This usually implies authentication and authorization of the request. As there is no password required for the Taxi Ordering application, signing the data using a key, which is known only by the client and the server, and is unique for each client mobile application, does this.

Figure 4 depicts the integration between the client mobile application and the server. The use case described in the sequence diagram is the one for Send Order functionality. On The client we have the *MVC* pattern. The view sends an event to the controller, which calls the send order operation on the Model. The model component responsible for sending the request (implemented in *Titanium* using an *HTTP Socket*), serializes the data and sends them to the specified service endpoint exposed by the *API*. Once the request arrives on the server, the reverse process occurs as the *JS* data objects are translated into *CLR* objects by the *WCF* framework. The security component is responsible for checking the signature of the user in the case of Order Taxi applications. In other applications a username and password approach can be used for obtaining the same result. It is essential to observe tat once the request passes the security verification, it is recorded and the registration key for the order is returned to the client. Further on the client uses this registration key for referring to his order. At this point there is no response available for the client yet.

The order placed by the user in the system is assigned to the nearest driver. When the driver confirms the order, a response is sent to the client mobile application. This response can come after a variable amount of time which can be anywhere between a few seconds and 3 minutes. In consequence SignalR is used to inform the client when the response is ready. SignalR is not used to pass the response directly to the user, but rather the user applications download the response once it receives the update signal.

An important characteristic of the service is that it should offer scalability. All the components depicted can be hosted in the *Cloud.*[4] This allows an elastic hosting environment, which can be used easily. For Order Taxi application, we are using *Windows Azure*. Using *Windows Azure* we can easily deploy the server component as it was built completely on the .NET framework. Though *Windows Azure* can also be used with many other development languages.

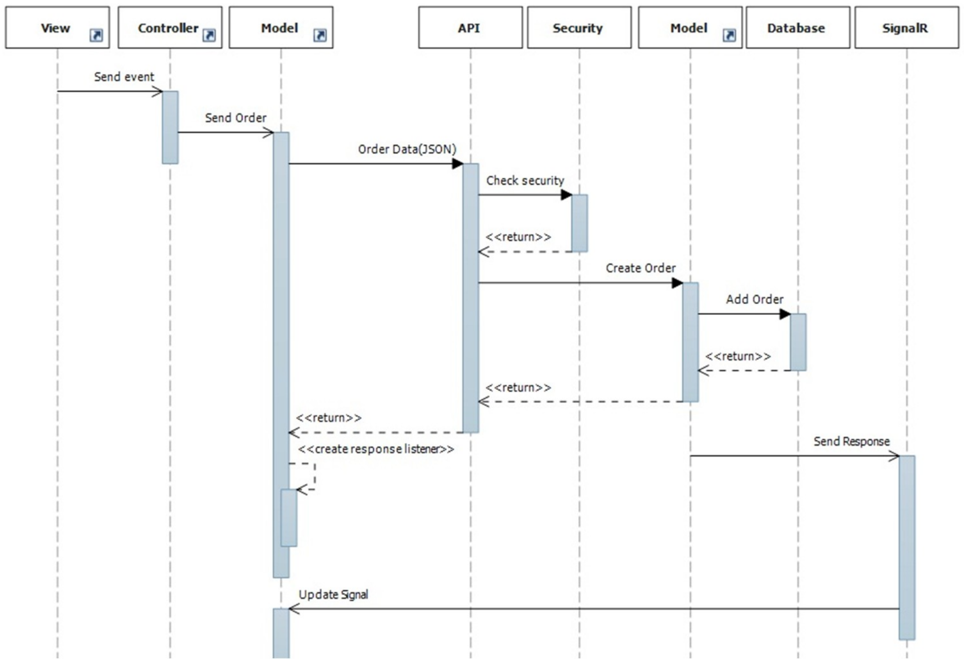


Figure 5: Integration of the client with the server API. Example for ordersending

3.2.Analytics

Analytics provide the means of measuring user experience inside the app and becomes a key component of the mobile ecosystem. Analytic provide the means of observing how users are interacting with the app and this provides the information required for further updates. Traditionally two analytics were used: *daily active users* and *user session time*. However it has been observed that user engagement falls sharply few days after the app first installation. As a consequence another key metric has become the user retention level after one day, one week and one month. Unless the app uses analytics it cannot understand its users base accordingly.
It is possible to devise ones own analytics system. However his is a time consuming and also challenging activity. For example analytics data need to be stored and only uploaded to the server when the user has an available Internet connection. However there are free services that provide analytics available. One of these services is *Flurry*, which provides a *REST API* for interaction. Activation of the API requires preregistration for their service for obtaining an *API* key. Titanium provides a free module, which can be used to encapsulate communication with the *Flurry* server.

3.3.Social networks integration

Social networking services are used in mobile applications in two ways. First they provide a mean of authentication using *OAuth.* For example if the app requires the users email address it can prompt the user to authenticate in the app using his *Facebook* account. If the user agrees to allow *Facebook* to share the email address with the app, the app is authorized to retrieve the users email from the Facebook service using the *Facebook* API. *OAuth* is an open standard for authorization. It allows users to share their private resources stored on one site with another site without having to hand out their credentials, typically supplying username and password tokens instead. This mechanism uses tokens for allowing authorization for specific resources and within a specific time frame. *OAuth* providers are not limited to social networking sites. *Google*, *Yahoo*, *AOL*, *Twitter, Flick, Youtube, Bitly* and many others offer OAuth services.

The other common use of *social services* integration is for marketing purposes. Many apps allow users to post messages to *Facebook* or to post on the users behalf. The posted messages should be based on users intention to share them, but they also often serve as a marketing strategy inside the users social network. *Order Taxi* application uses *social services* for marketing purposes by sharing the users ride information with the users consent.

In order to use social services, as with most third party *APIs* an *API* key must be obtained by registering on the providers website. For *Facebook* this is available in the developers section of their website. Titanium offers modules for integrating *Facebook* and *Twitter* services and encapsulating the communication with the *REST API*.

3.4.Third party services integration

There are a vast number of 3rd party services available. We have covered previously some of the most common used in the majority of apps. However depending on the app target market other services need to be considered. Their integration inside the system is performed in the same way as the previously discussed ones, be means of a *REST API* in the majority of cases. For example *Order Taxi App* uses *Google Reverse Geocoding* service for converting *GPS* coordinates into street address information. Also The SMS gateway system is used for sending SMS to the mobile for registration.
Other services common with mobile apps are:

1) *QR code readers*

QR codes are used to store information. Types of information stored in QR codes include text, phone number, URLs and SMS messages. There are numerous API that can process a QR code photo and extract the information from an image.

2) *Payment processors*

Mobile payment systems tend to be localized and fees differences between regions might make one credit card processor better suited than other. Titanium does provide a module for payment using PayPal. One convenient solution without implementing the API is using a WebView control in Titanium for displaying the pay by credit card webpage of your desired credit card processor.

3) *Advertising*

Many apps rely on advertising for generating income, especially if they have a high user base and long usage sessions, such as games. There are numerous advertising providers and there are also Titanium modules built for the most common, including Google AdSense.

4) *Context awareness*

Some apps require location related information given the users position. Urban Airship provides such a service after acquiring SimpleGeo. Foursqare and Yelp can also be used to get specific information for a GPS position as well.

5) *Speech recognition and keyword spotting*

Such services can be used to allow a voice interface for the app. Speech recognition is concerned about converting speech to voice while keyword spotting tries to identyfy certain keywords.

4. Testing

There are three types of components that need to be managed: The proprietary server, the mobile client and the 3rd party services. Managing the 3rd party services is usually done by a management panel on the providers website. A management panel for changes that have a high occurrence rate manages the proprietary server. For example new drivers are added manually to the system from this management panel. The hardest component to be managed is the mobile client application as there is no direct control over it. If major changes to the system are required for implementing new functionalities and this requires a change in the API it is important that API versioning is used to secure reliability for old clients that do not install the last updates. Backward compatibility of the system must always be respected. If the changes made to the system do not require any change in the API, but rater a change in the address, this can be resolved if the client was designed to adapt to such changes. Otherwise an update of the client app is required.

Usability testing of software applications developed for mobile devices faces a variety of challenges due to unique features of mobile devices, such as limited bandwidth, unreliability of wireless networks, as well as the changing context. For assuring the system correctness Use Case Driven Testing of the functionalities of the system must be addressed in various environmental contexts such as no Internet access, no GPS signal etc.

Order Taxi app was designed with these considerations in mind and the testing approach was by using Use Case Driven testing in various conditions. The following issues have been tested:

4.1.Mobile context

It can be defined as “any information that characterizes a situation related to the interaction between users, applications, and the surrounding environment [10].” For our Order Taxi app we have tested how the app was performing in different times and places, especially how GPS was acting inside buildings and how well wireless networks aided the positioning based on the accuracy of position determination. It is very difficult to select a methodology that can include all possibilities of mobile context in a single usability test[10]. If the user position cannot be obtained and the GPS was turned off, we asked the user to turn it on or input the address manually.

4.2.Conectivity

Lack of connectivity or slow Internet speed is common with mobile applications. We tested whether the app can identify this situation and alert it to the user, notifying him that the app can not work properly and he has to either turn on internet (if this is not turn on already), or just report that the app can not work in the absence of an internet connection.

4.3.Screen size and resolution fragmentation

These concerns affect mobile applications. We tested the applications on multiple devices and also used UI elements that can adapt proportionally to screen sizes.

4.4.Lack of storage and computing power

These requirements must be tested especially if the work is not mitigated to the Cloud. For our app this was not a concerned as we both outsourced the processing and storage to the Cloud.

Another component than must be tested is the API of the proprietary server. The test cases should include unit testing and security testing. We used a test project that generated a set of test cases periodically that the API had to pass in order to verify both the security and the business rules implementation.

5. Related work

5.1.Mobile development

HTML5 is regarded as being the future of Internet and also most of the apps, which are now built natively, are expected to be available as *HTML5*. However *HTML5* specifications, which are developed by World Wide Web Consortium, are not available yet and the release target for specifications has been delayed until 2014. While there are already a number of apps available which work based on a subset of *HTML5* specifications, the majority of top apps are still native, especially those which require a high integration with the platform. *PhoneGap* is a cross platform framework that rivals titanium but operates differently. While the result of building an app with *PhoneGap* is a native app, this app is basically a wrapper around an *HTML5* page, which exposes a JavaScript interface for interacting with specific device features such as camera. We recommended using Titanium as a cross platform solution based on the large number of apps that have been built on the platform (over 40000), the native capabilities which go deeper than *PhoneGap* as it doesn't rely on a *WebView* and also the possibility to target both native mobile and *HTML5* releases.

5.2.Service development

NodeJS is a software system designed for writing *WebServers*. InfoWorld has designated *NodeJS* as "Technology of the year 2012". The main difference Node.js brings is that it is event based and not thread-based. *NodeJS* has been especially credited for allowing easy communication between the client browser and the server. *Socket.IO* is a client side *JavaScript* library that talks to *Node.js*. *Nowjs* is a library that lets you call the client from the server. All these and SignalR are similar and related, but different perspectives on the same concepts. *SignalR* is a complete client and server-side solution with *JS* (JavaScript)on client and *ASP.NET* on the back end to create these kinds of applications. *SignalR* brings the possibility on developing using *C#* on the server, which we considered to be a benefit at the present time as the *.NET* framework is more mature in terms of development environment, productivity, security and libraries. *NodeJS* has the advantage of flattening the development stack as it uses *JavaScript* on the server side, and thus it is possible to develop a full *Mobile Cloud Computing* system using only *JavaScript*.

6. Further developments

3rd party services

Third party services are appearing every day and their capabilities become even more impressive. In our example application we can further integrate other *3rd party services* and expand its functionality. It is important to monitor new entrants to the market in order to offer the user the most complete and updated system in terms of desired functionality.

Security

Considering our implementation of the security on the *API* layer, we can observe that using a hash function algorithm on a key, which should ideally be known by only the client app and the server, does the user authorization. However this key is sent trough *SMS* which cannot be considered a secure channel. As a solution the key could be sent encrypted or part of the key could be used using another mechanism and that combined on the client. Sending the key encrypted trough *SMS* diminishes the usability of the system, as a 4-digit code can get very complicated. As a solution the SMS could be read inside the app where it could be decrypted. SSL is used for all communication between the client and the server, using a self-signed certificate.

7. Conclusions

We have presented the building blocks of a complete *Mobile Cloud Computing* system and discussed the vast array of possible technologies that can be selected. We have motivated the choice we made for using *Titanium* compared to other cross platform frameworks such as *PhoneGap* and *HTML5* or compared to native development. On the cloud part we explained how this is integrated with the mobile client and with other third party services and how specific functionalities for mobile clients such as Push Notifications can be triggered from the server.

We also examined trending technologies for implementing web servers such as *NodeJS* and we presented how *SignalR* can be used in conjunction with *.NET* framework for providing the same advantages as *async* servers. We also classified the most important 3rd party services and how this services can integrate with the client mobile application and the server. Mobile clients present a new set of challenges in terms of usability given the precise interaction they require on a small screen so we presented the usability tests which were performed on the Order Taxi application which was the example used along this paper for illustrating the concepts.

References

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. Web Services: Concepts, Architectures, Applications. Springer, 2004.
[2] L. Bass, P. Clements, and R. Kazman,Software Architecture in Practice. Addison Wesley, 2003.
[3] J. Tyree and A. Akerman. "Architecture decisions: Demystifying architecture". *IEEE Software*, 2005.
[4] White Paper, "Mobile Cloud Computing Solution Brief," AEPONA, November 2010.
[5] Hoang T. Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. A Survey of Mobile Cloud Computing: Architecture, Applications, and Approaches, Wiley, 2011
[6] P.Zou, C.Wang, Z.Liu, and D.Bao, "Phosphor: A Cloud Based DRMS cheme with Sim Card," in Proceedings of the 12th International Annual Symposium on VLSI, August 2002.
[7] A. Smailagicand M. Ettus, "System Designand Power Optimization for Mobile Computers, " in Proceedings of IEEE Computer Society Asia-Pacific on Web Conference (APWEB), June 2010.
[8] Cesare Pautasso, Olaf Zimmermann, Frank Leymann. RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision, 2010.
[9] J. Oberheide, K. Veeraraghavan, E. Cooke, J. Flinn, and F. Jahanian. "Virtualized in-cloud security services for mobile devices," in *Proceedings of the 1st Workshop on Virtualization in Mobile Computing (MobiVirt)*, June 2008
[10] Anind K. Dey and Gregory D. Abowd, Towards a Better Understanding of Context and Context-Awareness, 2001