Evolving Mobile Architectures: A Case Study in the Development of a Location Privacy Application

Jeffry Babb
jbabb@mail.wtamu.edu

Kareem Dana
kdana@mail.wtamu.edu

Department of Computer Information and Decision Management
West Texas A&M University
Canyon, TX 79016

Mark Keith
mark.keith@gmail.com
Department of Information Systems
Brigham Young University
Provo, UT 84602

Musa Jafar
mjafar@mail.wtamu.edu
Department of Computer Information and Decision Management
West Texas A&M University
Canyon, TX 79016

Abstract

The use of mobile devices, and the applications that run on them, has soared in recent years. Among the reasons for this rapid uptake is the inclusion of many useful sensors (including GPS, accelerometers, and cameras), the plethora of mobile apps, and improved battery life. These same advances in the capabilities of mobile devices and applications can also lead to privacy concerns; particularly those related to location privacy. We developed Find-a-mine, an iPhone application (and supporting infrastructure) to conduct privacy research through a scavenger-hunt style mobile application-based game. This paper presents a retrospective on mobile device trends especially within the context of location-based privacy, describes the design and development of the Find-a-mine application ecosystem, discusses the development challenges faced, and provides our thoughts regarding the future development of this application and more generally, mobile applications as a whole.

Keywords: Location Privacy, Location-based services, Mobile applications, Mobile Architectures, Mobile Computing

1. INTRODUCTION

Interest in mobile application development has risen steadily with the latest generation of mobile devices. We can define this current generation of devices as being those whose capabilities are extended by: (1) the presence of a sophisticated and dedicated operating system (Android, iOS, etc.); (2) the presence of additional hardware features such as sensors (GPS, accelerometer, camera, etc) and telecommunications transceivers (802.11 wireless, bluetooth, IMT-2000/3G, etc.); (3) a mature API and specification for the development of independent applications which target the mobile devices; (4) intuitive and ergonomic HCI interface which involved extensive haptic and tactile interaction. While the common referent for this generation of mobile device is the "smart phone," the same architecture has been extended to other form-factors, most notably the "tablet," which should not be confused with earlier stylus-oriented tablet PCs.

While these features have created a new innovation milestone and epoch, the attendant ancillary issues related to use – those related to societal, organizational, and personal use – are emergent and worthy of investigation. One area of use that is of increasing interest to researchers and users alike is that of location privacy. Although the literature on location-based services extends further into the past than does the current generation of mobile devices, the nature of these services has been transformed by the confluence of the capabilities of the new devices and the uptake in the usage of these devices. Much as was the case with computer viruses, the true impact of location-based privacy issues will be-come clearer as the total number of users in-creases.

This paper describes a mobile-application architecture developed to facilitate a research project designed to determine self-efficacy beliefs related to mobile application use and subject of privacy in the consumption of location-based services. The name of the mobile computing ecosystem and artifact which resulted from this development effort is Find-a-mine. With the Find-a-mine application, we have developed a scavenger-hunt style interaction "game," where participants both answer attitudinal and affective questions related to their self-efficacy in using the mobile devices and also play a game where participants are willing to test the boundaries of their location-privacy tolerance. This mobile application architecture was developed in 2010 and 2011 and has become a valuable learning experience with respect to best practices and pitfalls.

This paper is a report on our experiences in writing this application and provides lessons learned and suggestions for improvement for both future iterations of our own application and perhaps ideas for others embarking on such a venture. The rest of the paper proceeds as follows. First, we discuss a brief background on mobile application development and expound on the aspects that define the current generation of mobile computing. Next, we share the implementation details of our mobile application and include the challenges faced and lessons learned during development. We next describe our plans for future iterations of the application based on our own lessons learned and recommendations. Last, we conclude with a final discussion.

2. BACKGROUND

The most impactful and poignant of contemporary mobile devices is the smart phone. With 55% market share (and rising) of mobile devices as of mid-2012 (Gold, 2012), the smartphone represents greater utility in convergence of features that were previously only available in separate devices. While the uptake of mobile phone usage has been on the rise for over ten years, there are sets of previously disparate features that have converged to make the latest generation of smartphones more transformative than was the case in previous generations (Ladd et al., 2010). The convergence of networking, data story, applications, and sensors into the same hand-held mobile device has transformed the living habits of many. The ability for people to stay connected, regardless of location, is increasingly accepted as expected condition of contemporary life (Pinchot et al., 2011). Among the advantages of the convergence of these technologies is the ability for the sensors on the device to assist the smartphone user in establishing the relevance of their current location to the user's information needs. Given the mobility of the smartphone, the ease with which a number of location-based services can be utilized has undoubtedly influenced the uptake of the latest generation of smartphones.

The Importance of Convergence

As recently as ten years ago, mobile phones were commonly just that, devices which enabled the user to access voice data networking, such as the Advanced Mobile Phone System (APMS/1G) or the Global System for Mobile Communications (GSM/2G) network, which offered few features beyond voice and texting (SMS). While these devices provided important voice and text services, other mobile computing devices, also popular at that time, were portable digital music players, Global Positioning System (GPS) receivers, and Personal Data Assistant (PDA) devices. Each of these devices would provide a set of enhancements for users for both entertainment and productivity purposes. However, intercommunication amongst these devices was rare, and rarely done in real time (Nusca, 2009).

The next wave of mobile phone technology gave rise to question whether the term "phone" was even appropriate as the device increasingly became a mobile computing platform upon which voice and text was among the many features. Early smartphones, such as the once-ubiquitous BlackBerry, some Nokia Phones, and the earliest iPhones, combined the functions of a personal digital assistant (PDA) and a mobile phone or camera phone. As subsequence versions of Apple's iPhone emerged, the smartphone was increasingly defined by the convergence of other features and combined the functions of portable media players, low-end compact digital cameras, pocket video cameras, and GPS navigation units (Nusca, 2009).

Developing Software for Smartphones

A final importance of the transition to smartphones as mobile computing platforms, and perhaps the defining aspect of the most current generation of smartphones, tablets, "phablets," and "superphones," is the presence of an underlying operating system for which 3-rd party applications can be developed against a well-tested and published application programming interface (API). Among the most widely-used and adopted APIs for smartphone development are the software development kits for Apple's iOS operating system and Google's Android operating system (Segan, 2012).

Although each of the major smartphone vendors now offers an extensive API for development of applications, as was the case with the emergence of the personal computer, developing applications across these platforms is hardly standardized or unified. For developers, this presents a level of fragmentation was common in supporting the multiple PC platforms (such as IBM PCs, Atari Amigas, Apple Macintosh, Commodore 64/128, etc.).

Just as was often the case in the emergence of the personal computing platforms, it is likely that many of the mobile computing devices are using similar hardware. However, despite these similarities, each mobile operating system, SDK, and development tools have enough differences to confound and complicate the application development process (Blom et al., 2008; Gavalas & Economou, 2011). It is for this reason, that we find mobile application developers face a similar set of development issues that have been encountered in previous eras of computing innovation (Kautz et al., 2007).

However, it is also clear the convergence of new technologies inherent to smartphone mobile computing brings new development challenges not previously experienced. For instance the array of sensors available as program inputs, the haptic interfaces, the small form factor, and the need to critically observe and preserve battery usage are fairly unique to smartphone mobile computing application development (Carroll & Heiser, 2010; Ferreira et al., 2011). Furthermore, user interfacing and usage of mobile applications follow patterns that are unique to smartphones as opposed to other less-mobile computing platforms, including laptops and even PDAs. Among the mobile computing use patterns most salient to this paper is the matter of location privacy in the use of location-based services.

Location Privacy

Whereas the current generation of smartphones provides an unprecedented level of mobile computing power and connectivity, there are issues related to the increased use of these devices that may not be fully understood by either the consumers adopting these devices, or the researchers interested in the intended and unintended consequences of the use of smartphones. Among these consequences of use that is salient to this paper is that of location privacy in the face of the consumption of location-based services. It is possible that, much as was the case with email and computer

viruses, smartphone users will not entirely understand the consequences of using a device which is able to juxtapose the user's location, in real time, against a plethora of invited, and uninvited, programs also utilizing the device (Keith et al., 2011).

If vendor information is taken at face value, the smartphone mobile computing platforms each have extensive built-in controls and features that should prevent loss of data or privacy as a result of malicious code. However, as smartphone are fundamentally based upon a mobile computing platform, they are vulnerable to attacks. Several security vulnerabilities and weaknesses have been identified recently, and it is possible that these, and other unforeseen weaknesses may be exploited such that a user's location privacy is compromised (Barkuss & Dey, 2003; Seriot, 2010). In this sense, a user's location privacy can be thought of as the degree to which information about a user's location during device use, and the context of using the device at that location, remains within the control and discretion of the user. Location privacy is compromised when the user's location information is divulged to another party without either the user's consent or awareness, or both. Even in the case that the vendor does provide a nearly flawless security implementations, in the case of the iPhone, many users circumvent these protections by "jailbreaking" the device out of its native operating parameters by either replacing the OS or altering it to unlock features and otherwise customize their experience. In any case, these new computing devices are both the source of great benefit and appreciable harm, depending on the circumstances of use.

The importance of this background information is to set the context in which the Find-a-mine mobile application for the iPhone, described in detail below, was developed. While the principle concern for writing this application was to facilitate mobile privacy research, we have also learned lessons regarding mobile application development practices and pitfalls. The purpose of this paper is not to offer lessons regarding the protection of location privacy, but rather offer insights pertinent to designing an application which utilizes the compelling features of the current generation of smartphone mobile applications: the utilization of an ecosystem of downloadable applications, a device that is persistently in touch with its surrounding information and physical environment (while powered on), a device which concentrates and converges a rich collection of information about a user and the user's environment.

3. DESIGN AND METHODOLOGY

On the surface, Find-a-mine is a mobile scavenger hunt game. Participants sign up on the game's web site (http://www.findamine.mobi) which explains the research nature of the application, what data will be tracked, and acts as a hub for users to manage their accounts and interact with other players. Currently, we are only recruiting university students to take part; however in the future, we believe this game can be expanded to the general public. To incentivize students to participate, gift cards are available to those who successfully complete the hunt. Winners are those who score the most points. Points can be award in a variety of ways including solving the most hunts or solving the hunts the fastest. After signing up, the participant logs in with the iPhone application to start the scavenger hunt. Figure 1 illustrates the login screen.

Figure 1 Login screen

While playing, a series of clues guide the player to various locations around their university campus. A Hot/Cold bar appears at the bottom of the screen providing feedback to the player as he or she is searching for the location hinted to by the clue as shown by Figure 2. The GPS capability on the iPhone determines if the player correctly found the location and can proceed to the next clue. Usually a single hunt will consist of about five clues with the final clue being the location of the "treasure" (an object or point of interest around campus). Once the final clue is reached, the player must take a picture of the

treasure. To accomplish the privacy research goals, the player's location is tracked throughout the entire game and the player is required to answer a set of research questions after each clue. Figure 3 shows what the research questions view looks like to the participant.

Figure 2 Playing the game

Figure 3 Research Questions View

Location data, the photo of the treasure, and research question answers all need to be accessible by the researchers in near real-time. In addition, the participants require access to the latest clues, available games, and leaderboard. Our design consisted of a three-tier architecture using a variety of different technologies as illustrated above by Figure 4. The iPhone application was developed using Objective-C and the XCode IDE. This is the standard iOS development environment supported by Apple. The web service was programmed in C# with Windows Communication Foundation (WCF) and acted as the middleware facilitating communication between the iPhone application and the database backend. The choice to use WCF was largely due to fact that the website was developed using ASP.Net, the same framework WCF is based on. Utilizing the same framework for all server-side development created a more seamless implementation and made development and maintenance easier. Furthermore, WCF was a new addition to the .NET framework and we believed it would be a useful API to learn. Finally, we utilized Microsoft SQL Server as the backend to store the data. By using technologies from the same developer, SQL Server integrated well with WCF.

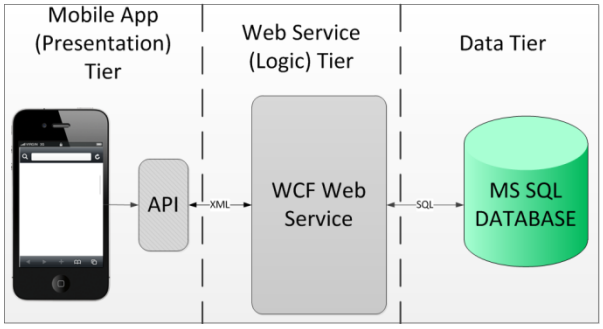

Figure 4: Architecture Overview

Early on we decided to develop for the iPhone platform first. Due to our limited resources, this decision meant that we could not concurrently develop for the Android platform. Android applications are programmed in Java while iPhone applications are programmed in Objective-C. Two completely separate code bases would need to be developed and maintained to write a single app for both platforms. We believe this is a significant problem facing mobile application developers; this is a problem we address towards the end of this paper.

The WCF web service communicates with the iPhone application through an application programming interface or API. Messages are passed back and forth in XML. This design allows for a future Android version to simply be plugged into the current architecture without changing the other components. It also had the benefit of compartmentalizing the development. The developers were geographically separated

during most of the development process– across Washington D.C., Tuscaloosa, Alabama, and Canyon, TX. We communicated via Skype and used Subversion for version control, but the majority of the development was done independently with one developer working on the iPhone application, one programming the WCF service, and one deploying the MS SQL backend and website. We were able to combine the three parts quickly near the end of development because of the initial API design.

Design Evaluation

Our design evolved over several iterations through discussions and testing before settling on the design described above. Even our current design presents some challenges that we discuss near the end of this paper as we look forward to future implementations. However, there were design challenges that we faced and solved early on in the design process. To evaluate our design, we utilized a small group of beta testers (students and other professors) that had very little knowledge of the app beforehand. These testers, in addition to the developers' field testing the app themselves, provided important feedback that was used to improve the usability of the application.

Figure 2 shows the main view of the app while a player is playing the game. There is a map with clue text on the top of the screen and a Found It! button in the bottom right corner. A small blue dot with an accuracy circle around it tracks the user's location as they travel towards the clue. Once a player is "close" to the clue, he or she can click the Found It! button to advance to the next clue. Testers had the most frustration with this window. They felt that the Found It! button was too sensitive. Oftentimes they would be near the clue but still not be able to advance to the next clue and had to reposition their phone or move a few meters in one direction or another before the Found It! button would acknowledge that they had reached the clue. This was and still is a very legitimate problem but a subtle one, not with the UI of the application but with the sensitivity of the GPS sensors.

We initially programmed the Found It! button to require the user to be within ten meters of the clue. This proved to be too strict of a requirement. For indoor clues, the iPhone GPS receiver rarely picks up an accurate GPS signal, so is oftentimes limited to WiFi or cell tower triangulation. Older devices such as the iPhone 3G and 3GS have less accurate GPS receivers, and through our testing we found out these devices are generally limited to a maximum of thirty meter accuracy. Also through beta testing we noticed that sometimes a device would lose the GPS signal for a brief period of time before reacquiring it. Moreover, the app did not provide any feedback to the user about how close they were to the clue or how accurate the GPS signal was. These issues were the main cause of frustration for our initial testers.

We implemented the Hot/Cold bar as a result of this feedback. The Hot/Cold bar received very positive feedback and alleviated a lot of user frustration. Users now had feedback from the app about how close they were to the clue and were not at the whim of a potentially fickle GPS signal. The other change we made as a result of our design evaluation was to increase the Found It! radius from ten meters to thirty meters and all the way to sixty-five meters if no GPS signal was available. These changes, only discoverable through actual testing of the design, helped solve many issues that the beta testers voiced.

4. IMPLEMENTATION ISSUES

While implementing Find-a-mine, we encountered several interesting challenges that we believe are unique to mobile development and worth discussion. First, mobile applications are gathering and utilizing more and more data. Data usage will only increase as apps become more complex and bandwidth becomes more available (Choi et al., 2011). Find-a-mine is no exception. Second, battery life remains a concern for mobile devices. It is prudent for mobile app developers to minimize power consumption especially if the app utilizes several sensors like GPS such as Find-a-mine.

Mobile Data Consistency and Management

Find-a-mine stores a lot of data. When accurately synced, the GPS sensor can generate up to one new latitude/longitude pair per second. In addition to that location data, answers to research questions, and photos must be stored. Apple provides the CoreData library for local data management. However, there does not yet exist a single, generic, and unified library to manage data across all tiers for mobile apps. Such a library should meet at least the following requirements:

- Ability to communicate with any RESTful web service keeping the middleware decoupled from the mobile app.
- On iOS, use CoreData locally to maintain compatibility and standards.
- Maintain consistency between local data and remote data.
- Minimize WiFi or 3G connection usage while syncing data to conserve power.
- Gracefully handle situations where no data connection is available.

We implemented a solution from scratch that met some of these requirements but was specific to our application's needs. To prevent the WiFi or 3G connection from being constantly utilized, we bundled location updates locally using CoreData. Once at least one hundred updates were accumulated, the application would upload them as a single bundle. Data was also uploaded on application exit or when the application entered the background. This ensured that all the data would be uploaded within a reasonable amount of time but also not consume power unnecessarily. If no data connection is available, the application simply periodically retries while maintaining state locally about what data still needs to be synced.

Since implementing our solution, several iOS libraries have emerged that attempt to tackle this challenge in a more complete, generic way. They are at varying levels of maturity and include RESTKit, RESTful CoreData, CoreResource, and ObjectiveResource.

Smart GPS Usage

Using the GPS radios on the iPhone require more power than any other operation on the device. Find-a-mine requires the use of GPS to track the user's location and also allow the user to complete the scavenger hunt successfully. The GPS is only turned on while the participant is actively playing the game, however, at the highest level of accuracy the GPS radios will significantly drain the battery (Abdesslem et al., 2009; Lin et al., 2010). We needed a smarter solution.

We developed an algorithm that gradually increases GPS resolution over time thereby reducing power consumption while still benefiting from the most accurate GPS readings when the participant was close to the clue. Using Apple's iOS Location Services API, an application is able to access six degrees of GPS resolution. Based on our testing during development, we encountered a range of accuracy readings from three kilometers, the lowest degree of GPS resolution, all the way to the highest-level accuracy, which we noted as generally within five meters. Our GPS resolution algorithm works as follows:

1. Start at one kilometer accuracy. Through testing, we found that starting at three kilometer accuracy (the least accurate GPS resolution available) provided no noticeable benefits and the algorithm quickly requested an increased resolution.
2. Determine the user's location at this level accuracy.
3. Calculate the distance between the user and the clue solution. iOS has a built-in "distanceFrom" function that takes two latitude, longitude pairs and will return the distance between them in meters using the great-circle distance.
4. If the distance between the user and the clue is less than the resolution accuracy, that is if the distance between the user and the clue is less than one kilometer and our accuracy is at one kilometer, then increase the GPS resolution by another degree.
5. Repeat steps 2 through 4 until the highest level GPS resolution is achieved.
6. Maintain the highest GPS resolution until the user finds the clue or exits/backgrounds the app.
7. Restart from Step 1 for each clue.

You can notice from this description that the application only requests the highest level GPS accuracy once the user is already very close to the clue. The user must be getting closer to the clue before the accuracy is raised. The application spends less time using the higher powered sensors and only uses them when it is necessary. In addition to improving battery life, this iterative process did not negatively affect the accuracy of the location results or the game play experience during our field testing. (Lin et al., 2010) implemented a similar solution of utilizing less accurate but less power hungry sensors and confirmed significant battery life improvements. SenseLess (Abdesslem et al., 2009) took advantage of the accelerometer along with the GPS to conserve power and achieved similar results to our own.

Fragmentation

As stated earlier, due to the fragmented nature of the mobile device market and our limited resources, we only developed an iPhone version of Find-a-mine. Solutions to this fragmentation problem have started to appear and are maturing. These solutions include Appcelerator, PhoneGap, Appspresso, and AppFurnace among others. These programs allow a developer to write an app once using JavaScript and compile both an iPhone and Android version from the single JavaScript code base. Had we used Appcelerator, Find-a-mine would have been cross-platform from the beginning. This is a very appealing choice for many developers, including us. These JavaScript based mobile architectures have some problems. The quality of development tools is still lacking and they often require native code modules to access specific hardware (Charland & Leroux, 2011). However, they do produce cross-platform code in a fraction of the time otherwise and are actively maturing. We anticipate these types of solutions will keep improving in the months and years to come.

Simplifying Web Services

After delving into WCF to write our RESTful web-service middleware, we realized the configuration necessary for the web service to function properly was very extensive and rather confusing, negating much of the benefit the WCF API offered over an open source alternative such as PHP, Python, or Ruby.

5. TOWARDS FUTURE ITERATIONS

The difficulties encountered and our lessons learned have informed our plans for subsequent iterations of our architecture. Primarily, we intend to address the following identified issues: application fragmentation related to multiple platforms; simplification of web services; and the simplification and scalability of the web and data backend.

Addressing Fragmentation

Appcelerator Titanium is a platform for developing mobile applications using JavaScript, and other web-based technologies, which will afford an opportunity to ameliorate the costs associated with maintaining native code for several platforms (at least for Android and iOS) as it will afford one codebase. It is remarkable that we are in this position again as languages and platforms such as Java and Python were designed to solve these problems. However, the market penetration and demand for iOS-based devices, and perhaps BlackBerry and Windows, would be hard to ignore if the desire is to open an app to the widest possible market. As we have selected Appcelerator, we are excluding smartphone platforms other than Android and iOS as Appcelerator will only targets Android and iOS. The biggest drawback to the Appcelerator approach is that Apple's SDK still requires that a Mac OSX is used in order to access the iOS SDK. This essentially forces the developer to opt for an Apple-based development environment. The implication of this limitation is that we must select the most costly development platform in order to develop for both platforms; a Windows or Linux development environment would have been less costly. As Appcelerator is presently available under many conditions at no cost, the cost of the Mac hardware is somewhat offset.

Simplification of Web Services

While Microsoft's WCF offers integration with ASP.NET and the power of the .NET platform, we did experience some configuration issues and overall increased complexity. RESTful web service implementations in either PHP or Python would offer two distinct advantages: 1) they are easier to configure and deploy and 2) hosting environments for these technologies are considerably less expensive. While implementing our infrastructure using Mircosoft's ASP.NET MVC is one solution to our first problem, commercially hosting any .NET application tends to be more expensive. By contrast, switching to a PHP-based solution (Zend, Symfony, CodeIgniter, Yii, or the Drupal Service module), a Python-based solution (Django or TurboGears), or a Ruby-based solution (Ruby on Rails, Sinatra) would be just as easy (or easier) to maintain, and considerably less expensive to host.

Web Platform

Another lesson learned pertains to our selection of backend implementation and our overall technology strategy for implementing the web application and service. While there is little doubt that ASP.NET is an extremely capable platform for hosting modern web applications and services, in many respects the overhead is quite high. Generally, the costs associated with hosting these services are high. Additionally,

ASP.NET can be complex to configure and integrate. While we have already discussed several open-source alternatives, it has become clearer that emerging cloud-based options might be better suited to not only our project, but to many mobile application projects.

Platform as a Service (PaaS) is an emerging concept useful for describing the deployment of the required application backend needs for many applications (Gonçalves & Ballon, 2011). However, we feel that utilizing PaaS would offer several advantages for mobile applications developers:

1. **Scalability**: Many mobile apps developers are likely involved in developing and provision a large number of mobile applications. Solutions such as Saleforce.com's Heroku, Amazon's Amazon Web Services, and Google's Google App Engine each provide all required elements (execution context for business-logic code, database storage and APIs, and various communication frameworks) at the exact capacity and scale required. App developers, providers, and vendors can provide the level of infrastructure support that is needed

2. **Portability**: Particularly with the open-source approaches mentioned (PHP, Python, or Ruby), many of the PaaS solutions offer reasonably interchangeable features and support. By contrast, while Microsoft's Azure does offer support for several open-source languages, ASP.NET is fairly limited to a reduced number of PaaS solutions.

3. **Integration**: The open-source oriented PaaS solutions offer very tight integration between data communication, data management, and business-logic components such that developers are not stitching together their backend technologies across disparate approaches.

4. **Collaboration**: While source control repositories such as Subversion and Git are abundant, a PaaS solution allows developers fairly liberal access to the deployment environment such that the development and deployment environments are tightly and seamlessly integrated.

5. **Multi-tenant**: The PaaS solutions are designed for hosting multiple applications, which would make supporting a backend environment for a variety of mobile applications much simpler.

While it is not certain that the next iteration of the Find-a-mine environment will move to a PaaS solution immediately, we do anticipate this move and have planned for it. The matter of migrating from ASP.NET to a PHP, Python, or Ruby-based solution is a complex matter and will take time. Moreover, other than the costs associated with using ASP.NET, there is no reason to expect that our existing ASP.NET-based web and backend solution can't be ported to a PaaS environment (such as Microsoft Azure hosting). In either case, both web and backend computing environments for mobile applications should be well served by a PaaS solution.

6. CONCLUSION

The purpose of this paper is to share architecture for an iPhone mobile application used to facilitate research regarding location privacy in the consumption of location-bases services. As a learning experience, we discuss the challenges and lessons learned as a result of developing and deploying our mobile application architecture, including both client and backend concerns. We developed Find-a-mine as an application which utilized a scavenger-hunt game in order to measure users' willingness to allow the sensors on their mobile devices to divulge their geospatial location. Our research design factored in the importance of actually building the application ourselves such that we are able to appreciate the issues involved.

We have learned that the code fragmentation issues attendant to supporting multiple platforms remains as problematic as it was in the days of supporting applications on the Personal Computer platforms. A tool like Appcelerator Titanium goes a long way to address the fragmentation issue, although they are difficult to solve entirely. Moreover, we learned that communication methods between mobile clients and web backend environments require care and planning. While we feel that utilizing RESTful services was a good choice, the selection of backend environment to facilitate these transactions can influence the overall

complexity of the development effort. While ASP.NET and WCF web services offer some advantages, more integrated alternatives may also be worth consideration.

Lastly, we anticipate that migrating to a Platform as a Service (PaaS) implementation for our web, data, and logic back-ends. This transition would offer greater flexibility and scalability moving forward. This approach will likely become increasingly common as it promises to offer more equity in access and services for a wider variety of mobile application ecosystems that also involve various backend services. This research has taught us that mobile applications and mobile devices are providing new experiences, challenges, and opportunities for users, developers, and researchers alike.

Acknowledgement

The software described in this paper was made possible, in part, through a research enhancement grant from the Kilgore Research Center at West Texas A & M University.

7. REFERENCES

Abdesslem, F. B., Phillips, A., & Henderson, T. (2009). Less is More: Energy-Efficient Mobile Sensing with SenseLess. *MobiHeld 2009: An ACM SIGCOMM 2009 workshop.* AUGUST 17, BARCELONA, SPAIN.

Appcelerator (2012). Retrieved 2012 29 June from Appclererator: http://http://www.appcelerator.com/

AppFurnace. (2012). Retrieved 2012 29 June from AppFurnacne: http://www.appfurnace.com/

Apple, Inc. (2012). Getting the User's Location: Tips for Conserving Battery Power. http://developer.apple.com. Retrieved 28 June 2012.

Appspresso (2012). Hybrid mobile app framework. Appspresso: http://appspresso.com/. Retrieved 28 June 2012.

Barkuss, L., & Dey, A. (2003). Location-Based Services and Mobile Telephony: a Study of Users' Privacy Concerns. *Proceedings of the INTERACT 2003, 9th IFIP TC13 International Conference on Human-Computer Interaction* (p. 4).

Blom, S., Book, M., Gruhn, V., Hrushchak, R., Kohler, A. (2008). Write Once, Run Anywhere A Survey of Mobile Runtime Environments. *The 3rd International Conference on Grid and Pervasive Computing Workshops, 2008. GPC Workshops '08*. 132-137.

Carroll, A., & Heiser, G. (2010). An Analysis of Power Consumption in a Smartphone. *Proceedings of the 2010 USENIX Annual Technical Conference*, p. 14.

Charland, A., & Leroux, B. (2011). Mobile Application Development: Web vs. Native. *ACM Queue*, 24 (5), 20-29.

Choi, K., Toh, K., & Byun, H. (2011). Realtime training on mobile devices for face recognition applications. *Pattern Recognition*, 44(2), 386-400.

Ferreira, D., Dey, A. K., & Kostakos, V. (2011). Understanding human-smartphone concerns: a study of battery life. *Proceedings of the 9th international conference on Pervasive Computing*. pp. 19-33

Gavalas, D. & Economou, D. Development Platforms for Mobile Applications: Status and Trends. *IEEE Software*, 28(1), 77-86.

Gold, J. (2012). Nielsen: Android nears 52% market share for US smartphones. Retrieved 13 July 2012, from NetworkWorld: http://www.networkworld.com/news/2012/071312-nielsen-android-260870.html

Kautz, K., Madsen, S., & Norbjerg, J. (2007). Persistent problems and practices in information systems development. *Information System Journal*, 17, 217-239.

Keith, M., Babb, J., Furner, C., & Abdullat, A. (2011). The Role of Mobile Self-Efficacy in the Adoption of Geospatially-Aware Applications: An Empirical Analysis of iPhone Users. *Proceedings of the 44th Hawaii International Conference on Systems Science*. Poipu, Hawaii.

Ladd, D. A., Datta, A., Sarker, S., & Yu, Y. (2010). Trends in Mobile Computing within

the IS Discipline: A Ten-Year Retrospective. *Communications of the Association for Information Systems*, 27(17), 285-360.

Lin, K., Kansal, A., Lymberopoulos, D., & Zhao, F. (2010). Energy-Accuracy Aware Localization for Mobile Devices. *MobiSys 10: PRoceedings of the 8th Annual International Conference on Mobile Systems, Applications and Services*. San Francisco, CA, USA.

Nusca, A. (2009, 08 20). Smartphone vs. feature phone arms race heats up; which did you buy? Retrieved 13 July, 2012, from ZDNet: http://www.zdnet.com/blog/gadgetreviews/smartphone-vs-feature-phone-arms-race-heats-up-which-did-you-buy/6836

PhoneGap. (n.d.). Retrieved 29 June, 2012 29-June from PhoneGap: http://www.phonegap.com

Pinchot, J., Paullet, K., & Rota, D. (2011). How Mobile Technology is Changing our Culture. *Journal of Information Systems Applied Research*, 4 (1), 39-48.

Purdy, G. (2010). Building a Server-Driven User Experience: Remote-controlled native UIs for fun and prof*it. Apple World-Wide Developers Conference 2010*. Presentation.

Segan, S. (2012). Enter the Phablet: A History of Phone-Tablet Hybrids. Retrieved 13 July, 2012, from PC Magazine: http://www.pcmag.com/slideshow/story/294004/enter-the-phablet-a-history-of-phone-tablet-hybrids

Seriot, N. (2010). iPhone Privacy. *Black Hat DC 2010*, Washington D.C. p. 30.