Ubiquitous Computing and Android

Zohaib Sibte Hassan
National University of Computer & Emerging Sciences, Lahore, Pakistan
zohaib.hassan@nu.edu.pk

Abstract

This paper discusses the properties of ubiquitous systems and brings to light the software requirements for such systems. Our study has shown that the primary principle behind all ubiquitous applications is to provide the users with intelligent learning systems that supplement their everyday activities. This is the primary consideration on which a ubiquitous application can be evaluated. This paper also discusses various features of Android operating system in the light of the ubiquitous system requirements. It also highlights how Android fulfills the subject requirements and considerations.

1. Introduction

Consider a world where desktop computers are replaced by embedded computers in typical physical objects, without interfering with the current functionality of those objects. That is what Weiser's vision was, and he termed it *ubiquitous computing*. The computers would be small enough to be embedded within the object and will provide enhancement to the functionality of object [3].
In this area, Xerox PARC was among the initial workers. They constructed computer prototypes in three different sizes .i.e. whiteboard, a pad of paper, and a post-it note size. The primary focus in this case was communication and context [1].
The goals for the PARC Tabs project were:

i. Design a mobile hardware device that enables personal communication.
ii. Design an architecture that enables mobile computing.
iii. Construct context sensitive application to exploit this architecture.
iv. Test this system in a community of people.

In *ubiquitous*, initially, the focus was on the small special-purpose devices, network protocols, etc. However, later new directions were identified in this context. The subsequent sections describe one of the efforts currently being carried out by Google in order to deliver a *ubiquitous* system on mobile phones. It is called Android.

In the present time, mobile phones are becoming an important tool with everyday utility, so much so that for some people it is the only mode of communication with other people. Mobile phones have the property of being a digital device equipped with reasonably good computing power. Such equipment, therefore, can be selected to serve as ubiquitous devices.
Following sections will discuss the specific requirements for ubiquitous devices, particularly for mobile phones. After that, the discussion on how Android is fitting those requirements, by providing a ubiquitous environment to programmers as well as users, will be undertaken.

2. Requirements

The requirements of a ubiquitous system have been classified in three categories: system, software, business and organization. System refers to the platform i.e. the hardware of the system, or the system running the subject software on top of it. The term software represents the services and the components of system that have been derived from the ubiquitous system. Business refers to the actors who are providing the components and services for the development of the system. Organization includes development methods and the processes needed for the development and integration of the system services [2]. This paper will only focus on the software requirements in the subsequent sections.

2.1. Interoperability

The ability of software to understand the information that it exchanges with other software or devices, and to provide something new based on the sent or received information is called interoperability [2]. This will enable the physical objects to operate in an intelligent and friendly manner.
In interoperability, there can be many hardware and software related issues and, therefore, limitations. e.g., what medium of communications are to be used and how to resolve two different set of protocols being used. The same problem can be mapped to

978-1-4244-2917-2/08/$25.00 ©2008 IEEE

various components when coming to software point of view.

2.2. Heterogeneity

For ubiquitous computing, the system is introduced to a wide range of devices belonging to different backgrounds and having different system and software configurations and requirements.
The diverse kinds of devices in the whole environment have varying interaction set-up, screen resolution, radio and memory capabilities, etc. Each of the devices can have its own style of interaction .e.g. via voice, touch screen, hand moment, buttons, etc. Therefore, to cater to such a diverse environment, a novel solution in the form of embedded sensor is required.

2.3. Mobility

An important characteristic of ubiquitous computing is mobility. However, the term *mobility* in its own has two kinds: actual mobility, and virtual and physical mobility. Since the devices are going to be interconnected via network, hence, mobility is a very important consideration.
Actual mobility is an extension to the capability of an autonomous software agent that can dynamically transfer itself to the node where the required resources are located.
When agents are aware of the distributed nature of the target, and explicitly locate and access the internet resources in the environment, this allows virtual mobility of agents across different execution environments.
Physical mobility means the connection of mobile and wireless computing devices to the internet from dynamically changing access point.

2.4. Survivability and Security

Survivability is ability of a system to fulfill its mission timely, and in the presence of attacks and failures. It requires a self-healing or repairing kind of infrastructure with qualities like security, performance, reliability, etc.
Security in an application may be implemented by various mechanisms. For example, login (but still be fragile by failing when the server or network dies). However, a surviving system must be able to survive malicious attacks by providing enhanced security mechanisms. [2] The survival can be of two types:

i. *Survival by Protection* [2]. In this type of survival the system protects itself from restricting the undesirable changes in the environment that can cause harm to the system.
ii. *Survival by Adoption* [2]. In this type of survival the system configures itself in such a way that adjusts with the respective environment having varying situations.

2.5. Adaptability

Software services have to handle dynamically emerging and evolving contexts and user preferences over different kinds of networks and terminals. Due to dynamically changing conditions, the adaptability gets into the bigger picture. Therefore, in a ubiquitous system it is also one of the main concerns. The adaptability approaches are delineated hereunder:

i. *Laissez-Faire* approach states that the responsibility of adaptation is completely left to the individual applications without any system support. It is more difficult to implement, as application size is large and it may not be possible in some cases for application to fit according to the system needs. Thus, may be problematic. But still being standalone, the application will possess all the power within it [2].
ii. *Application Transparent Approach* says that system should provide all the necessary support libraries etc. for making an application adaptable. This makes system responsible to handle everything. This will result in much smaller applications since the shared components are present in system [2].
iii. *Application-aware:* It is middle way approach in which application as well as the system contains the functionality [2].

2.6. Ability of Self-Organization

A system that has the ability of self organization will be able to increase its organization in context of the situation at hand. Such a system has the ability to create its own organization. A ubiquitous system should have the ability of self organization.

2.7. Augmented Reality and Scalable Content

In the present times, the study of augmented reality with nouvelle perspective is paving its way. It is one of the hottest issues in the context aware system; which is the core of a ubiquitous system. Data consistency and high availability of data is an important part of a ubiquitous system.

3. Challenges for Ubiquity in Mobile Phones

As already mentioned above, ubiquity can be applied in mobile phones, since they are a tool of daily usage.
This very characteristic enhances their importance. We can look at the mobile phones as wearable computers that one can carry around without hassle. This enables free movement and hence, more utility. There are various functionalities that a mobile phone offers, which includes the provision of accessing the internet, checking email, and downloading content from various sources, without the limitation of presence of a particular environment to perform all these tasks.
One might consider that a Personal Computer (PC) has the ability of performing all of the above tasks, so why not carry a PC along? This is because the actual problem exists in the way a human being has to interact with a PC. In order to perform a particular task on the computer the human being will have to learn the language that the computer understands. On the contrary, the actual goal should be to enable the computer to understand what the user desires.
So the core purpose behind it is that we do not want computing at all. We want information, and our system to be context aware so that it can deliver the information to us in timely manner automatically.
Putting it in simpler words, we want information to be aware of itself, aware of the user's situation and deliver itself in a timely manner.
A simple example could be of a user *x* who uses the Global Positioning System (GPS). He has the information about another person *y*, and if *y* comes across, *x* becomes aware of it with the help of the GPS information he has. In this scenario, it is extremely important that *x* is informed immediately and accurately about the presence and location of *y*, so that he can act accordingly. A ubiquitous mobile device is expected to display such intelligence. There are two other important properties that a ubiquitous device should have, they are discussed below.

3.1. Implicit Interaction

Implicit interaction takes place when a system is aware of the functionalities that a user is performing, but it doesn't, however, take decisions. The system actually follows the movements and choices of the user and presents the respective options. It is the responsibility of the user to select one of the listed options and then act accordingly.
This is an important and crucial part of a ubiquitous device, because the choice of the user can change dynamically and the system should act in a reactive manner for presenting the options.
In this scenario, several improvements can be made. A system can analyze the relative information that is likely to be useful to the user in a particular situation, and inform the user about that information. [1]
This will enable the user to make better choices for future decisions.

3.2. Task Based Interaction

Task based interaction means that the system should be aware of the environment that it is facing, and should be able to update its status accordingly.
Normally a user has to specify what computing requirements he desires, and use certain hardware and software utilities accordingly. In such a situation, the system is not aware of where it actually exists and therefore, cannot update its status according to the status of the user.
In a ubiquitous environment, it is extremely cumbersome for a user to manually update the status of a system with respect to the scenario it is facing. [1] Therefore, an important feature for a ubiquitous device to be able to revise its status according to different situations.

4. Android as a Ubiquitous System

In the following sections, we will be analyzing Android in the light of the ubiquitous computing features and characteristics discussed above.
At this point, it is important to take a look at the architecture of Android, as released officially by Google.

Figure 1: Android Operating System Architecture

Keeping the above architecture in mind, the following sections will analyze several features of Android.

4.1. Core of Android

The core of Android operating system is built on top of LINUX kernel, since the kernel has a proven driver model as well as security model with a lot of available drivers. So LINUX is serving as the hardware abstraction system in Android [5].

4.2. Interoperability in Android

Android provides a unique and a very generic way to support interoperability. From software point of view, Android provides a *content provider* [6] that is responsible of handling the published data. Each application can use the content provider to publish its data that it may want to share with other applications. Other applications can then ask the content provider to get this data. Thus, in component integration portion, the content provider has lowered the coupling of classes and application components. A simple example for content provider can be a phone book. As phone book numbers are required in most of the applications, the phone book can publish its contents by content provider making other applications independent of exactly what version or application is being used for phone book.

4.3. Heterogeneity in Android

Android operating system in itself, from middle-ware and the software layers above it, provides a heterogeneous interface to its users [5].

In middle-ware, libraries like *OpenGL ES*, *SGL*, *Surface Manager*, and *Free Type* provide a cross-platform layer that is only dependent on the LINUX layer below it, and it provides the layers above it a uniform interface.

For example, the *Surface Manager* that runs on top of *OpenGL ES* and *SGL* makes the user independent of what exactly are the VGA details of the device; thus allowing the user to talk only in terms of the surfaces. Similarly, the *Free Type* is meant for the user to give support for various fonts in system. Similarly, *SQLITE* which is an open-source and light weight embedded SQL engine provides highly optimized data storage support. *Web Kit* (as web browser rendering engine), and *LIBC* provide the upper layers a uniform interface regardless of what particular hardware is working under these layers.

The most important component of the system *Android Runtime* is just like *Java Runtime;* in fact, it is an optimized version of java runtime for mobile phones. The core of which lies in DALVIK virtual machine, this component can run something called DEX files. The DEX files are almost the same as JAR file. So no matter on which hardware the DEX file was compiled on, it can be executed on any device as long as it has all the package or library dependencies installed.

4.4. Mobility in Android

Since the operating system is designed for mobiles hence, the physical mobility is present. From the aspect of virtual or software mobility, since all the applications are based on virtual machine, it means that applications or agents can easily port themselves on any machine or device with Android on it and perform execution. *Activity Manager* is responsible for saving the state of an activity or an application state. Once this state has been saved by the system [5], it can be streamed or sent to other devices where it can be restored.

The important thing to be noticed is that *DALVIK VM* lies in the core middle ware of Android. This means that any application on upper layer won't be using the direct interfaces below. Which also means that any software that is required to run on an Android system; must be in DEX format. Hence, by giving an independent executable format for services or applications, the operating system in itself has made sure that the execution is cross-platform. This also implies that agent software can be designed to transfer and execute itself on any Android system where it has the required resources.

4.5. Survivability and Security in Android

Android has achieved the survivability by protection mechanism in itself, since the architecture is based on LINUX; which has over the time, proved itself to be among the league of secure operating systems. So at the bottom level *SELINUX* is taking care of operating system level security. However, the intercommunication security has been achieved by building in the SSL component [5]. By using more secure communication mechanisms the two devices communicating via any device driver in kernel can use the security mechanisms to ensure that there is no intrusion in the system. By restricting invalid or unknown intrusion from any entity the system has made itself secure at operating system level and communication level as well.

4.6. Adaptability in Android

The core architecture of Android has been designed on the principle of re-usage of components. Hence by replacing a component one can easily and seamlessly provide adaptable behavior to user.
For example, picking a photo from the system for an application like *GMAIL* mail client. In order to do that, the application or the process must make a request to system that it wants a picture as input from user. From the list of the installed applications Android selects the best available application to fill up the request (according to the user's preference), so assumption can be taken that system album viewer is available to full fill the request. Hence, its execution is started and request is full filled. Now take a scenario in which the mobile user does not want the built in album application and replaces it with some enhanced application for album manipulation like *PICASA*. Now the next time when the same request is made, Android will invoke *PICASA* instead of systems default album viewer. So according to user preference an application has changed the way it works.
This approach of Android has made other applications or components of the system less cohesive to other applications. In addition to this applications can carry their own specific components along with them as well. Just like pluggable JAR components the components for particular applications can be provided. Hence, system is using a *hybrid approach* to adaptability of application [5].

4.7. Self Organization and Augmented Reality in Android

Android is already running some applications demonstrating its ability to self organize and view representation of augmented reality. For example of *Google Maps* application, this application can communicate with its neighboring devices in order to get maps of a locality before trying to connect to internet and download the maps.
Local communication can be achieved by *XMPP Service*. With *XMPP Service* two or any number of devices can communicate with each other; Thus giving applications a very neat and clean communication mechanism.
Sensor based applications have not appeared yet, but the architecture seems to be more promising to in compensate such features.

4.8. Implicit Interaction in Android

Android provides a very versatile method of implicit interaction. Equipped with the *LINUX kernel* and powerful device driver support it can use any device; by simply installing its driver [5, 6]. Current available example of implicit interaction is via GPS and maps application. Since the user can observe the change in his position on map when he is driving a car or moving to someplace, the concept for implicit interaction is present in Android.

4.9. Task Based Interaction in Android

Notification Manager and *Activity Manager* in Android is responsible for providing user with task based interaction; it not only maintains the state of an activity, but also a common back stack to save the order of activities. So if a user has just opened a browser window from a mail application, Android maintains the record of what applications were open and in which order they were opened. This will give the user a smooth transition in the story board and give the same effect that work has been resumed from where it was left.
The notification manager is responsible for notifying any external event to a running service or application. So, it can take input from any of the input device available in the system, and update the asynchronously send notifications to multiple services or applications.

5. Discussion

After the release of iphone which is based on the information ubiquitous principle there are sudden changes in trends to use mobiles just not as mobiles but as devices that can provide you some information in a more friendly and interactive manner.
Google has made a great effort by establishing Open Handset Alliance (OHA) in which they are setting up hardware standards for next generation cell phones. In previous sections, the study of how the ubiquitous features and challenges are being mapped on the operating system architecture was undertaken. Following is a brief summary table of the discussion.

Table 1: Feature-wise Analysis of Android

Feature	Implementation In Android
Interoperability	*Content Provider* handles it
Heterogeneity	*Core libraries* and *Android Runtime* handle it. It also includes components like *window manager*, *view systems*, and *telephony manager* etc.
Mobility	*Activity Manager* handles it
Survivability	*Survival by protection* is achieved by Linux at core and SSL layer in middle-ware, giving an excellent security mechanism.

Adaptability	*Resource Manager* and *Package Manager* provide the system with information to encourage reusability of components.
Self-Organization	*Notification Manager, XMPP Service and Location Manager* take up the responsibility to make system self organized and context aware.
Augmented Reality and Scalable content	*OpenGL, Linux kernel and proven driver model of Linux* makes it possible to implement such applications
Implicit Interaction	*Location Manager, and Linux driver model* makes it possible for system to interact implicitly with user
Task-based Interaction	*Notification Manager, Activity Manager* makes task based interaction possible.

With the above set of features of Android it is most likely that architecture is going to be like benchmark architecture for other upcoming technologies. Some other frameworks like Qtopia and OpenMOKO also appeared early in the market but they were not able to catch as much attention as Android did. The basic reason may be the flexibility of the design and the usage of many powerful technologies. Each and every component that can be seen in the middle ware is a giant in its own, like OpenGL that's running industry wide and providing the ability to render 3D graphics on phone, similarly Web Kit that has been used as core of Safari a very well known browser, SQLITE a proven embedded database technology that is very light weight and powerful as well, Free Type a library for fonts and bitmaps rendering, SSL the standard on which today's websites are relying while any kind of secure transaction is being made, and on top of it they have provided a virtual machine that is almost same as that of java technology; in fact totally java technology customized for mobiles. These powerful bits and pieces have compiled together to one big promising giant.

6. Conclusion

There are tradeoffs in information awareness of computer; we definitely need to make some security and privacy analysis. The "aware information" term definitely makes computer more responsible for configuring and monitoring themselves, and determining when to deliver information. Thus, before moving and handing over private information to our devices we must have thorough analysis, work out on performance as well as security issues, only then one promising system can appear. As Weiser's article was concluded saying "*ubicomp is likely to provide a framework for interesting and productive work for many more years or decades, but we have much more to learn*" still remains unmodification.

7. References

[1] J. Scholz, "Ubiquitous Computing Goes Mobile", *Mobile Computing and Communications Review*, ACM Publications, Vol.5 Number 3.

[2] E. Niemela, J. Latvakoski, "Survey of Requirements and Solutions for Ubiquitous Software", *Mobile Ubiquitous Computing Conference*, ACM Publications, 2004.

[3] M. Wieser, "The Computing for the Twenty-First Century", *Scientific American*, 1991, pp. 94-104.

[4] B. Agarwalla, G. Abowd, Umakishore, Ramachandran, "Toward a Standard Ubiquitous Computing Framework", *2nd Workshop on Middleware for Pervasive and Ad-Hoc Computing,* ACM Publications, Toronto, Canada.

[5] Official android website, Google Inc., http://code.google.com/android/, 2008.

[6] You tube, Mike baron's videos on Androidology, Google Inc, http://www.youtube.com/user/androiddevelopers/, 2008.