Ubiquitous Computing – Concepts & Challenges

Dr. X. Joshphin Jasaline Anitha

***Abstract*— The terms Ubiquitous was first coined at the beginning of the 90's, by Xerox PARC and IBM respectively, and capture the realization that the computing focus was going to change from the PC to a more distributed, mobile and embedded form of computing. This paper will describe where software and hardware have combined to enable ubiquitous computing, where these systems have limitations and where the biggest challenges still remain.**

***Index Terms*— Ubiquitous Computing, Power Management, Wireless discovery, User Interface Adaptation, & Location Aware**

I. INTRODUCTION

Ubiquitous computing is a broad semantic definition. In many cases, researchers define ubiquitous computing in their research projects through examples. We use Weisers definition of ubiquitous computing. He defines ubiquitous computing as a phase in the development and use of computer systems in which they permeate our environment and are integrated in most artifacts, adding useful information services in an unobtrusive manner. i.e., ubiquitous computing is the method of enhancing computer use by making many computers available throughout the physical environment, but making them effectively invisible to the user.

The main future of ubiquitous computing is to create a user centric and application oriented computing environment. Such environments are different from the traditional computing models since a physical space within the environments supported by associated hardware and software facilitates interactive information exchange between users and the space. The availability of cheap computing devices and wireless networks are making such spaces possible. A user does not require to log into a single personal computer as in traditional computing environments, but communicates with a variety of computing devices in the space. Scalable configuration is an important aspect in such space since the same space is often used for different tasks at different times.

A. The evolving human-computer relationship

Internet, Internet2, intranets, extranets, cyberspace . . . it is hard not to have heard or read about one of these terms in the media. Several trends categorize computer use in the information era.

1. Mainframe stage: Computers were used by experts behind closed doors, and regarded as rare and expensive assets. This stage was the beginning of the information era. The human-computer relationship was one of several humans to a single computer.

2. Personal computing stage: In this stage the human computer relationship became balanced in the sense that individuals had one-on-one relationships with their computers. This stage brought certain closeness into the human-computer relationship.

3. Ubiquitous computing stage: In this stage one person will have many computers. People will have access to computers placed in their offices, walls, clothing, cars, planes, organs, etc. This stage will have a significant impact on society

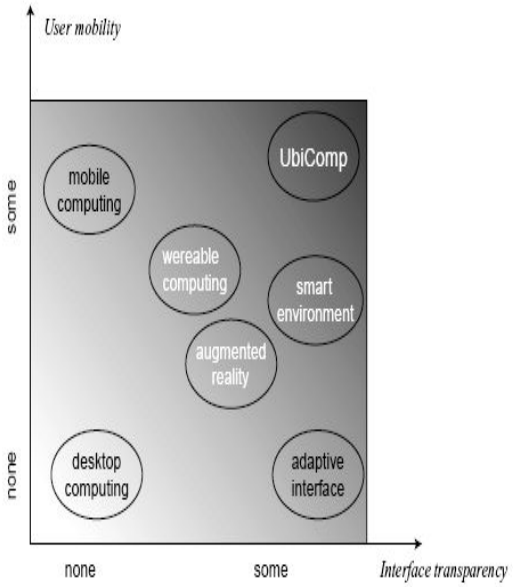
II. AN ONTOLOGICAL FRAMEWORK OF UC

Weiser's classification of a ubiquitous computing system is based on two fundamental attributes:

- ✓ **Ubiquity:** interaction with the system is available wherever the user needs it.
- ✓ **Transparency:** the system is non-intrusive and is integrated into the everyday environment.

According with this classification Abowd identified two dimensions that provide a rather clear boundary for ubiquitous computing systems and express the relationship with other emerging research areas:

- ✓ **User mobility:** reflect the freedom the user has to move about when interacting with the system.
- ✓ **Interface transparency:** applies to the system's interface and reflects the conscious effort end attention the system requires of the user, either for operating it or for perceiving its output.



An Ontological Framework

Manuscript received Feb 12, 2016

Dr. X. Joshphin Jasaline Anitha, HoD, Department of Master of Computer Applications , N M S S. Vellaichamy Nadar College , Nagamalai,Madurai, Tami Nadu, India

III. CHALLENGES FOR UBIQUITOUS COMPUTING

In the remainder of this paper we focus in more detail on the challenges for ubiquitous computing systems, the progress that has been made towards a solution, and the software engineering work that still needs to be done by our community. We will conclude this paper with a summary of the most salient issues that need attention in this area.

Facing the Challenges

There are many challenges facing the design of successful ubiquitous computing systems. Here we focus on four of the issues for which progress is being made and therefore represent a challenge worth undertaking power management, wireless discovery, user interface adaptation, and location aware computing.

A. Power management

The power consumption of Ubiquitous computing devices can be divided into three main components: ***processing, storage, and communication.*** For each of these categories, there is usually a technique for controlling power that involves "turning a knob" for a given component: reducing power by decreasing speed, range, or capacity by simply lowering duty cycle of the device. For example, it is quite easy to lower the power of a wireless transmitter by reducing the output power, effectively reducing its range. Alternatively, it can be highly effective to control power by switching between heterogeneous subsystems within a conceptual element of the system: e.g., to switch between radios technologies such as Bluetooth and WiFi for wireless communication. However, this technique, which can offer an order-of-magnitude improvement in power consumption, requires more software support to deal with the accompanying heterogeneous software interfaces to these systems.

Processing is a highly variable component for a ubiquitous computing platform – its requirements can very from very little for simple monitor-and-wait applications to extremely high for computationally intensive tasks such as running a neural net. Within a single processor, it is possible to control the power consumption by either selectively deactivating individual blocks, like the multiplier, when they are not in use or lowering the operating voltage to slow the part down and also reduce the energy per operation using a technique known as DVM. Beyond this, it can be highly effective to control power consumption by utilizing multiple kinds of processors within a system: for example, a full-function processor for the main computation, an embedded microcontroller for sensor monitoring, and a network processor for processing network packets. In fact, many systems already possess multiple processors, e.g. the firmware in a wireless card, but their functions are largely hidden, reducing their overall effectiveness. The software engineering challenge is figuring out a way to expose the processing components of individual subsystems for general use, allowing a system to "push down" some operations to the sub-processor when beneficial; furthermore, this would need to be done in a flexible manner, so that the software language used is not tied to the specific components, allowing a write-one run-anywhere policy for such hierarchical processor systems.

Wireless interfaces present a similar challenge to that of multiple processors. WiFi, Bluetooth, and Ultra Wide Band (UWB) are all either existing or up-and-coming radio standards with widely varying capabilities and characteristics. Each one has various power-control and performance settings, for example the basic transmit strength or listen duty cycle, but an order-of-magnitude performance gain is possible by utilizing multiple radios in one system because each technology is individually targeted at a specific usage model. For example, WiFi is suited for in-home wireless Internet browsing, while Bluetooth is better suited for low-power devices such as cell-phones. Often, the overall optimal solution will require combining multiple radio systems into one device: organizing them as a wireless hierarchy, that, for example, capitalizes on the low energy-per-bit of WiFi, but can rely on the low standby power of Bluetooth. For the implementation, each radio system encompasses different characteristics, connection model, and programming interface. The software engineering challenge is to provide a layer of abstraction that effectively offers the overall best service to the platform without unnecessarily complicating the higher-level interfaces. For example, a way to offer the power saving benefits of using Bluetooth within the context of an in-home WiFi network, even though a single Bluetooth node will not reach though an entire house.

Local storage is another sub-system that can consume considerable power in a Ubiquitous system. There are many different kinds of storage media that are available for such systems: physical disks, flash, RAM, etc… Similar to the processor- and wireless- subsystems, each kind of storage presents a different power profile to the system. Flash, for example, is very good for idle and read power, but is considerably lower density than a physical disk and is very slow for writing. Likewise, the software engineering challenge is to provide flexible access to storage capabilities without over complication. One concrete example is managing the power consumption of a full-fledged operating system like Linux in the embedded environment. Although it provides a great wealth of capabilities, its operational memory footprint will be considerably larger than a purely embedded operating system like TinyOS. This operational footprint is important because if it was small enough to fit in the system's available SRAM, the system could power-down its DRAM during an extended sleep operation while still offering quick-wakeup capability.

These three basic components all have great potential for power optimization at the considerable risk of overcomplicating the software interfaces. The reason systems like Java, Linux, and TCP/IP networking have become so popular is that they are uniform: software engineers don't need to do something different for different environments. True, they don't always live up to the *"write once, run anywhere"* mantra, but they at least offer a *"learn once, use anywhere"* environment, which greatly increases their potential. The challenge with these heterogeneous systems, which offer a complex array of capabilities and trade-offs, is to generate a similar uniform interface that programmers can utilize. There are obviously other considerations with power, as described briefly in the following section, but the ability to effectively manage the applications running on embedded platforms is a key enabling step.

B. limitations of wireless discovery

Another significant problem facing emerging Ubiquitous computing systems is the management of the many small

computing nodes comprising a large, complex system. As the number of computers per person increases, the conceptual, physical, and virtual management of these devices becomes a problem. Going forward, embedding processing in everyday objects, such as a coffee cup or chair, exasperates this problem: drastically increasing the number of computing devices that must somehow be managed.

One major problem facing any large collection of small devices is just a basic understanding of what exists: if we know something exists, how do we find it, or, if we have a collection of devices, how do we know what they are? Often, people have trouble keeping track of just their keys and cell-phones – imagine this problem on a grand scale where there are hundreds of devices around the house waiting to be lost, found, and eventually used! One solution to this problem is to release objects from a specific designation, and treat a television as just a television, instead of a specific television. This shift, which will make it easier to juggle a multitude of devices in the physical space, raises a challenge for software systems that now must be able to interact with many devices that have no unique individual address or identity. Typically, computer systems are addressed by a unique name, or IP address – a convenience that just may not exist in a deeply embedded environment.

Several emerging technologies, such as UPnP (Universal Plug and Play Device Architecture) specifically aim to make it easier to manage multiple devices in the home environment. For example, they aim to make it easy to bring home a new device, such as a scanner or home theatre, and hook it up to your home network: You bring it home, plug it in, and use your desktop PC or smart TV to coordinate its actions with other devices. By utilizing the infrastructure supplied by a coherent home network and desktop PC, this system makes it easy to connect devices – however, requiring manual connection and configuration would quickly become onerous for a large number of very small devices. So, although these technologies work for individual devices that can be recognized and handled by people, it is not clear how they will adapt to the challenges outlined in the previous paragraph.

At a basic level, Ubiquitous devices will not always be plugged into a wired network, requiring integrated wireless discovery techniques that raise significant questions about network integration. One significant problem in this space is the basic neighbor's printer problem: how do you integrate a new wireless printer into your home network without accidentally incorporating your neighbor's printer, or giving your neighbor access to your device. Basically, wireless networks are virtual in their physical topology – it is easy for them to co-exist in the same space while presenting a different logical construct to the user – a problem that is not nearly as common with wired networks. One solution is to require physical interaction with your new device, maybe temporarily plugging in a USB cable to initially configure the setup, or maybe you assume/hope that your neighbor bought a different model printer than you, making it easy to discern. But in the end, this simple scenario of bringing home a new device with "easy to use" wireless networking raises many fundamental challenges about how these devices are connected and associated. Now, just imagine this problem for a hundred small embedded computing devices!

The shift from single devices with well-known names and easy to discern network connections to a multiplicity of semi-anonymous objects arbitrarily connected to other nearby objects, presents a significant challenge for both software engineering and the basic supporting technologies. Systems will need to be mode intelligent and adaptable, automatically figuring out which devices are appropriate for any given set of interactions, and which devices "belong" in a particular space. Of course, on top of this there is also the quintessential problem of power management: how can you find and replace all the batteries needed to power the multitude of devices in an environment! Solutions to these problems must balance ease of use, privacy, security, cost, maintainability, and any number of additional constraints, making them anything but trivial.

C. user interface adaptation

A characteristic of Ubiquitous Computing systems is that they integrate a wide variety of devices from very small sensors, to palm, notebook and workstation computers, each with very different display sizes. From a system designer's point of view, applications need to operate effectively in this heterogeneous environment, and the users must be able to gain control of each component unencumbered by the physical difficultly imposed by size. For example, small devices imply small displays, and even the best UI design at this scale requires the user to navigate a series of terse menus. The problem is illustrated by typical experiences with PDAs, often loaded with features but never used because they are buried in the complexity of the interface. Some time peoples may face some problems about beaming a phone number between two Palm devices using an Infrared link, and in the end giving up, instead resorting to writing the number on a post-it note, and sticking on the recipient's PDA.

When users need access to information contained in a computer, the most effective interfaces are those well adapted to people. Long before computers existed many mechanical and informational tools were honed based on this principle, for example, books can be manufactured at any scale, but paperbacks are the convenient size they are because they are optimized for readability and portability. When building small devices for Ubiquitous computing applications, a means to adapt the interface to a more person oriented size can make the difference necessary to cross the usability threshold. Consider the cell phone; perhaps the most successful Ubiquitous Computing device to date, but limited by its size, the display is also constrained. But by using a local wireless connection to a more capable device, it is possible to access data on the phone using a familiar web browser interface on a full-sized desktop display.

To take full advantage of this capability, applications written for use with the phone display need to be able to adapt to a larger display when it becomes available. There are also occasions when information flow from applications written for servers with large displays would like to shrink their output for display on a smaller device. This situation occurs today with WAP based phones that wish to access WWW content providers such as yahoo.com. Current approaches detect the type of device in use and connect to a server capable of generating the required graphic elements on the small display, but to make use of this model the content for the smaller display size needs to be individually crafted.

A more flexible approach would allow application writers to generate UIs based on an abstract definition of the user interface and in combination with knowledge of the capabilities of the target display, generate the user interface components on the fly. Four components are needed to build such a system, a 1) user interface specification language 2) two-way control protocol 3) appliance adaptors and 4) the graphic user interface generator. Despite the success of this work it is a hard to create the building blocks that will result in widespread use and be adopted by product designers as a standard.

For any software product, the user interface is the one piece of the system that is placed in full view of the customers, and may make or break the business depending on its usability. For dynamic UIs, designers would be uncertain of how their application will manifest itself on the various screen sizes used by their customers. Often, the lowest common denominator ends up defining the result; however, software tools that clarify the result across a common range of target platforms can mitigate this issue. The up side has great potential as the richer the display, the richer the automatically generated UI, and potentially the better the user experience. In Ubiquitous Computing environments, the range of target screen sizes is far greater than typically found in the PC world; therefore, if the software products continue to work in these environments, the market size and potential revenues will be considerably larger. However, significant software engineering hurdles still remain in creating standards and the basic mechanisms to generate and display content to the user.

D. Location aware computing

One of the distinguishing features of Ubiquitous Computing over conventional distributed computing is the use of location to augment the data services available. Unlike the Internet in which a server is typically unaware of the location of the client, when computing becomes embedded into the local environment, interaction can be customized to improve the result. For example, a query about the multimedia equipment nearby can be automatically qualified by the current location and return information that is relevant to that room, rather than all the facilities available in the building. Likewise information accessed at a particular location on one occasion can be remembered and potentially offered to others who are about to make similar queries, thus opening up options that some people may have been unaware of.

The use of location context goes beyond just knowing where you are, but can take into account 'who' is with you, further building contextual clues about the activity undertaken at that time. A system supporting a conference room application might automatically provide electronic links on an electronic whiteboard referencing all the documents that were accessed by that group on the previous occasion they met. Likewise, messages with a low priority might not be delivered with an audible notification to a laptop, if the system is aware that other people are nearby and a meeting is likely to be in progress.

In some universities, experimental context-aware toolkits have been built to facilitate the design of context-based systems. However these tools have not seen widespread adoption. One possible reason is the uncertainty of location estimates: it may not be possible for a system to know exactly where something is, so how does it describe the range of possible locations? To address these concerns, research at the University of Washington created the Location Stack as a means of working with several sources of location information at once, and fusing the data together in a way that improved the overall accuracy and allows applications to understand the error distribution involved with the location estimate.

There are clear indications that location-based data enables valuable applications. Some mobile service providers' service has a feature to allow its clients to make location queries such as finding the nearest restaurants, based on location data inherent to the current cell tower in use. The most successful search engine, Google, has recently added local search to their search engine, allowing a query to be qualified by location. At present, a user is requested to type in a qualifying string for the location, but this service is ripe for extensions based on GPS and other location automation technology. Successful adoption of these services in the metropolitan area may well provide motivation for system designers to use these techniques at the local level, in much the same way that global web searches, are being complimented with Google Desktop in order to have a similar service run on the file system of a personal machine.

To date, the tools for location-based applications have not been available on mass to the industry. Although we now know how to turn wireless infrastructure into location-finding systems, these mechanisms have not filtered into the standard system building tools. A challenge for the community is to take these techniques and turn them into location-based APIs at the platform level, which can then be accessed on a wide variety of devices and utilize a variety of location technologies.

IV. CONCLUSION

Ubiquitous Computing were in some senses a way of taming real environments by placing embedded computation directly in the artifacts that are needed for physical work, and using orchestrated wireless communication to build a fully integrated system, with greater user value through augmentation. In this short paper we have focused on issues that are directly related to Ubiquitous Computing, but it should also be pointed out that many of the problems associated with distributed systems in general are also embodied in this area. In ubiquitous computing the applications are most likely aimed at diverse work practices with the computation hidden from view, and thus unless the maintenance of these systems can be automated the deployment will remain limited to domains in which the help from an expert is readily available.

V. REFERENCES

[1] Weiser, M., The Computer for the 21st Century, Scientific American Ubicomp paper, September 1991, (http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html)

[2] Weiser, M., Seely Brown, J., Designing Calm Technology, Xerox PARC, December 21, 1995, (http://www.ubiq.com/hypertext/weiser/calmtech/calmtech.htm)

[3] Microsoft. Understanding UPnP: A White Paper. 2000. http://www.upnp.org/download/UPNP_UnderstandingUPNP.doc,
[4] Want. R.; Schilit, B.; Adams, N.; Gold, R.; Goldberg, D.; Petersen, K.; Ellis, J.; Weiser, M., "An Overview of the Parctab Ubiquitous Computing Experiment", IEEE Personal Communications, Vol 2. No.6, 1995.
[5] Weiser, M. "Some Computer Science Problems in Ubiquitous Computing". Communications of the ACM, July 1993.

BIOGRAPHIE

With 17 years of teaching experience in computer science, She has done her research in the area of software Reengineering. She completed her M.Phil from Mother Theresa University, Kodaikanal. She did her Undergraduate in B.Sc(Mathematics) in Jeyaraj annapackiam college for women, Periyakulam. She started her career as Lecturer in Computer Science. She never confined her self to teaching alone. She is also passionate in motivating the youth especially the student community to become ,first and foremost,good citizens of the country. She is also equally interested in imparting creativity, self confidence and helping tendency among the students. At present,she is working as the Head of the deparment of MCA,at N.M.S.S.V.N College ,Madurai .She has been rendering her service there since 1999. She has also guided PG and M.Phil., projects of students from various Universities.