System Challenges for Ubiquitous & Pervasive Computing

Roy Want
Intel Research
2200 Mission College Blvd.
Santa Clara, CA 95052
roy.want@intel.com

Trevor Pering
Intel Research
2200 Mission College Blvd.
Santa Clara, CA 95052
trevor.pering@intel.com

ABSTRACT
The terms Ubiquitous and Pervasive computing were 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. Furthermore, it was predicted by some researchers that the true value of embedded computing would come from the orchestration of the various computational components into a much richer and adaptable system than had previously been possible. Now some fifteen years later, we have made progress towards these aims. The Hardware platforms used to implement these systems encapsulate significant computation capability in a small form-factor, consume little power and have a small cost. However, the system software capabilities have not advanced at a pace that can take full advantage of this infrastructure. 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.

Categories and Subject Descriptors
C.3 [**Special-purpose and Application-based Systems**]: Real-time and Embedded Systems, Microprocessor/microcomputer applications, Smartcards.

General Terms
Design, Management, Reliability, Standardization

Keywords
Ubiquitous & Pervasive Computing, Power Management, Wireless discovery, User Interface Adaptation, Context-Aware.

1. INTRODUCTION
Many of the tasks that are undertaken in everyday work practices can be augmented by some form of computation; however, beyond the digital office applications, such as world processing and spreadsheets, it is much more difficult to understand how to apply computation in a form that is readily appropriate for the task. For instance, a nurse may find that the traditional pen & clipboard approach to taking notes while talking with a patient may be better suited to their interaction than turning away to type at a conventional PC, despite the resulting inefficiency of needing to re-enter it into an archival electronic form later on. The ideals of Ubiquitous Computing, a term first used by Mark Weiser [16] at Xerox PARC in 1988, attempt to understand how to integrate computation with the physical world in a way that blends in so completely that it becomes unnoticeable, a property often referred to as "invisibility". One of his analogues used to convey the idea is our interaction with ink and the printed word: if a page is well presented we rarely dwell on the printing technology, the ink or the font, but instead are rapidly drawn into the information that is being conveyed. However, current computer systems are rarely this transparent and often force the users to dwell on issues that are purely related to the processing platform, not the application that is being supported. Faults are rarely simple to diagnose and any system that involves several cooperating computers usually requires specialist knowledge for its set-up before any work can begin.

Fifteen years ago, some of the challenges for Ubiquitous Computing were more at the component level than at the system level. The performance available in PCs was determined by the capabilities of the Intel 486 processor, while low-power microcontrollers were only good enough for 'washing machine' class applications and crude handheld devices. Embedded processors lacked the horsepower, or peripheral support, that is needed to solve real world problems. Also, wireless networking, a critical component for interconnecting the mobile components of a ubiquitous system was in its infancy and there were no widely adopted wireless standards.

Early attempts to integrate computing into a more mobile and wirelessly connected environment such as Xerox's Parctab, the first palm sized context-aware computer [14], were based on system-level design principles that allowed researchers to 'time travel' into a future use model, but traded-off system communication for computation and storage. For example, the Parctab system allowed complex applications to be run in remote workstations: the mobile devices were designed as wireless 'dumb' displays, sending pen and button events through a diffuse infrared network, while responses were received as graphic updates for the display. An additional benefit of this computational offloading was that the power budget in the mobile devices became manageable, and they remained small. Currently, the important difference between these heavily networked systems and those using today's technologies is that the trade-off between communication, processing, and storage in these systems has shifted and is continuing to shift in favor of more computation and storage in the mobile system components, allowing lower interaction latency and more resilience to disconnected operation. However, software engineering techniques have not as yet adapted to offer an easy way for developers to balance these considerations, highlighting an area for future progress.

Now with Intel XScale class processors available at clock rates reaching upwards of 0.6GHz, flash memory subsystems exceeding 1GB, and power budgets of only 500mW, it becomes practical to run complex calculations locally and only use the communication channel to capture ephemeral data, control other systems, to send messages, or to convey the results of a calculation. Furthermore, the growth of the storage component is exponentially outstripping the bandwidth improvements in the communication channel. Thus from a pragmatic software engineering perspective, preloading content in mobile devices will become the favored design approach. At this time, it is still important to carefully consider what needs to be preloaded, but as the exponential storage curve continues upward, everything that might be needed while mobile, will be preloaded as a matter of course and the algorithms can become less selective.

Although systems have continually been improving, power has always been a thorn in the side of the Ubiquitous computing vision. Since computation as we know it is based on electronic devices, Ubiquitous computing also implies the ubiquitous need for power. For fixed infrastructure, wired power suffices, and it is the general reliability of the commercial power grid that has fuelled the electric and electronic product economy. Batteries, induction, or energy scavenging from the local environment can provide power for mobile devices; but chemical-based battery technology continues to be the optimal choice. If you have smoke detectors in your home, which are now inexpensive enough to put in most rooms, you will go through the yearly ritual of changing batteries to ensure reliability, and will immediately resonate with the problems facing pervasive computing. For many mobile or embedded applications, such as sensor systems, power can be conserved through judicious control of its active operation. However building distributed systems using conventional software tools leaves these issues unaddressed, and the power consequences for creating software abstractions or using higher-level languages such as Java, may not be exposed to the programmers.

As each of the basic components required for Ubiquitous computing have fallen into place, the limitations on success have moved to the higher levels of abstraction. Low power radio technologies and common standards exist, but individual standards typically address a specific usage model and, when taken individually, do not typically present the best overall solution. Radio hierarchies [8], which aggregate the capabilities of any number of individual solutions, offer a more complete and advanced capability, but will require software engineering work to effectively abstract the idiosyncrasies of the various underlying technologies to present a uniform interface for applications. At the basic level, this technique consists of understanding how the low-power Bluetooth standard could augment standard WiFi network operation, offering a lower-power solution with the same long-range and high-bandwidth capabilities.

Another component that plays a crucial role in the evolution of ubiquitous systems is the display. Early portable LCD displays, although low power, were monochrome and low resolution. It was hard to meet the ideals of device 'invisibility' when the medium used for information transfer was well below the necessary readability threshold. Displays have steadily increased in quality: Color is now standard, and the resolution well above that used by television – a metric for large-scale acceptability. Quality displays are even appearing on inexpensive cell phones as the current ~600 million units annual sale is driving down cost, and driving-up technical competition on a feature-by-feature basis. However, applications are still written with a particular display format in mind and do not have the ability to transform their layout from one set of display dimensions to another. As wireless capabilities become more prevalent on all devices, the use of a device's display may no longer be limited to the device it is physically attached to. A computer with an impoverished display can now consider broadcasting its display pixels across a wireless link to a computer with better screen quality. The challenge presented here is how to build design tools that can abstract away the size of the display at the time an application is written.

Location technologies were an integral part of the vision of early Ubiquitous Computing systems. At that time unique location finding technologies, such as the Active Badge [15] had to be devised in order to provide in-building location services. However, with the advent of standards such as 802.11b/g location, to some reasonable level of accuracy, can be determined using the underlying properties of the wireless system [1]. Even Bluetooth provides this capability by the very nature that it is a short-range wireless technology, in most implementations not extending more than 12 feet. Ambiguities about what is close, due to walls and other physical partitions, can be removed if a comprehensive model of the environment is known ahead of time (sometimes called the *world model*). The use of location and other environmental knowledge in ubiquitous systems is usually referred to as being *context-aware*, and holds promise to simplify the interfaces we present to our mobile users, adapting them to the current situation. However, most computerized organizational support systems do not take advantage of location, and this presents an opportunity for future system designers.

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.

2. 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.

2.1 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 or an operating parameter 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 radio 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 [7]. 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, Zigbee, 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* [8], 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, DRAM, SRAM integrated with the processor, 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. Similarly, SRAM is very low power but at a fraction of the density of DRAM. 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 [5]. 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 (i.e., without having to reboot).

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.

2.2 Limitations of wireless discovery

Another significant problem facing emerging Ubiquitous and Pervasive computing systems is the management of the many small computing nodes comprising a large, complex system. In the early days of computers, there were many people using one computer; then, with the PC revolution, we reached an era of one person relating to one computer; now, starting with the introduction of cell-phones and other portable electronic devices, we now have multiple computing devices associated with each person. 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 I know something exists, how do I find it, or, if I have a collection of devices, how do I 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 coffee cup as just *a* coffee cup, instead of a *specific* coffee cup. 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, IP address, or MAC address – a convenience that just may not exist in a deeply embedded environment.

Several emerging technologies, such as UPnP [12] and Rendezvous [10], 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 printer or stereo component, 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 and/or Pervasive 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.

2.3 User Interface Adaptation

A characteristic of Ubiquitous Computing systems is that they integrate a wide variety of devices from very tiny 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, tiny devices imply tiny 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. A colleague recently recounted an anecdote about beaming a phone number between two Palm devices (while under some time pressure) 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.

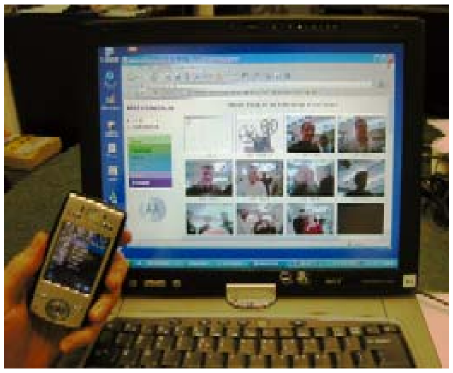

Figure 1: UI sharing between a cell phone and a laptop computer, across a short range wireless link.

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 (and the trend to keep building them smaller), 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. For example, in an extension of Intel's Personal Server project [13] in figure 1, the photos taken by the phone can be browsed and selected on a large display for transfer to the PC, though a wireless link.

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. In an exploratory toolkit called PUC [6], four components are needed to build such a system, a 1) user interface specification language 2) two-way control protocol, providing an abstracted communication channel between the application and user interface, 3) appliance adaptors, allowing the control protocol to be translated into the primitives available on the target device, and 4) the graphic user interface generator. Other toolkit examples are SUPPLE [3] and iCrafter [9]. 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. These problems are similar to those facing the designers of websites requiring the presentation to look good on a variety of browser's screens. 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 (the HTML equivalent for adaptable UIs) and the basic mechanisms to generate and display content to the user.

2.4 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 universities, experimental context-aware toolkits [2] 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 [4] 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. ATT's M-mode 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 (assuming they know it), 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.

3. Conclusion

Many of the visions of computer science such as artificial intelligence, image recognition, and natural language interaction, have proved extremely difficult to achieve. The impressive calculations performed by a computer are far removed from the kind of open-ended problems presented by real world situations, and are not readily applied to the solution.

Ubiquitous and Pervasive 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. However, even this simplified model of the world presents difficult problems that are not easily resolved. Early on the issues related to the inadequacies of fundamental components such as displays, low-power processors, high-capacity memories or wireless standards. More recently the limitations are associated with high-level system issues, such as power management, the co-ordination of wireless services, application user interface adaptation or the creation of generic capabilities to allow developers access to contextual information.

For example, power management continues to be a problem for ubiquitous systems that involve mobility, or utilize disconnected operation, which by contrast is not an issue for infrastructural computing. Many energy conservation techniques are now well understood and improve the battery lifetime of mobile systems, but power management still needs careful consideration. Although embedded system designers know these techniques, the tools to allow them to be used by system software architects at a high level of abstraction, through conventional programming libraries, are not available. As with all the solutions we find in the ubiquitous computing arena, until they are integrated into standard tools and widely adopted, we will not see the impact that could be achieved.

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. The design of distributed systems is still hard and when multiplied by a larger number of heterogeneous computers connected by wireless networks become commensurately more complex. Just as in distributed systems, achieving the desired result when all components are fully operational is not necessarily difficult, but when failures occur automating the diagnosis and recovering from the fault is a much bigger problem, and usually requires the users to take some part in the process. However, in ubiquitous and pervasive 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.

ACKNOWLEDGMENTS

The authors wish to thank Intel Corporation for its support.

REFERENCES

[1] Bahl, P.; and Padmanabhan, V. N.,"*RADAR: An In-Building RF-based User Location and Tracking System*" IEEE Infocom 2000, volume 2, pages 775-784, March. 2000

[2] Dey, A., K.; Salber, D.; and Abowd, G., D., "*A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications*", Human-Computer Interaction (HCI) Journal, Vol. 16, 2001

[3] Gajos, K., Weld, D. "*SUPPLE: Automatically Generating User Interfaces*", in Intelligent User Interfaces (IUI) '04. Funcha , Portugal, 2004.

[4] Hightower, J.; Brumitt, B., and Borriello, G., "*The Location Stack: A Layered Model for Location in Ubiquitous Computing,*" in Proceedings of the 4th IEEE Workshop on Mobile Computing Systems & Applications (WMCSA 2002), (Callicoon, NY), pp. 22-28, June 2002.

[5] Levis, P.; Madden, S.; Gay, D.; Polastre, J.; Szewczyk, R.; Woo, A; Brewer, E.; and Culler, D., "*The Emergence of Networking Abstractions and Techniques in TinyOS*", Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation, 2004.

[6] Nichols, J.; Myers, B., A.; and Litwack, K., "*Improving Automatic Interface Generation with Smart Templates*", Intelligent User Interfaces (IUI) Funchal, Portugal. pp 13-16, Jan. 2004.

[7] Pering, T.; Burd, T.; and Brodersen, R., W., "*Voltage scheduling in the lpARM microprocessor system*" In Proceedings of the 2000 International Symposium on Low Power Electronics and Design, pages 96-101, July 2000

[8] Pering. T.; Raghunathan, V.; Want, R. , "*Exploiting radio hierarchies for power-efficient wireless discovery and connect setup*", In Proc. of the 18th International Conference on VLSI Design, January 2005.

[9] Ponnekanti, S., R.; Lee, B.; Fox, A.; Pat Hanrahan, and Terry Winograd, "*ICrafter: A Service Framework for Ubiquitous Computing Environments*." In Ubicomp 2001, pages 56-75. Georgia, Atlanta, September-October 2001.

[10] Rendezvous, Apple's automatic discovery mechanism for computers, devices, and services on an IP network. http://developer.apple.com/macosx/rendezvous/

[11] Schilit, B.; Adams, N.; Want, R., "*Context-Aware Computing Applications*", 1st Annual Workshop on Mobile Computing Systems and Applications (WMCSA), Santa Cruz, Dec 1994.

[12] UPnP, "*Understanding Universal Plug and Play*", Microsoft white paper available at http://www.upnp.org

[13] Want, R.; Pering, T.; Danneels, G.; Kumar, M; Sundar, M.; and Light, J., "*The Personal Server: changing the way we think about ubiquitous computing*", Proceedings of Ubicomp 2002: 4th International Conference on Ubiquitous Computing, Springer LNCS 2498, Goteborg, Sweden, pp194-209, Sept 30th-Oct 2nd, 2002.

[14] 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, pp28-43 December 1995.

[15] Want, R.; Hopper A.; Falcao, V.; Gibbons, J., "*The Active Badge Location System*", ACM Transactions on Office Information Systems, vol. 10. No. 1, pp91-102, Jan 1992.

[16] Weiser, M., "*The Computer for the 21st Century*", Scientific American, Vol. 265 No. 3, pp94-100, September 1991