Evaluation of context distribution methods via Bluetooth and WLAN: Insights gained while examining Battery Power Consumption

Alisa Devlic
Appear Networks Systems AB &
Royal Institute of Technology (KTH)
Stockholm, Sweden
devlic@kth.se

Alan Graf
Ericsson AB
Tellusborgsvägen 83-87
Stockholm, Sweden
alan.graf@ericsson.com

Paolo Barone
HP Innovation Center
Via Grande 4
Milan, Italy
paolo.barone@hp.com

ABSTRACT
In a traditional context-aware system, most context information is local to a device. However, we may need access to context information from outside the device. Increasingly mobile electronic devices are equipped with Bluetooth and/or WLAN network interfaces. Both of these technologies enable ad hoc discovery & networking. In this paper we evaluate the use of these technologies for context distribution within a local area (i.e., limited to a single hop). Using Bluetooth, we begin by discovering devices using Bluetooth's discovery protocol, collect their context information, create an XML file containing this information, and distribute this file to all discovered devices, such that every device now has the same context information. Next we perform the same discovery, collect, and distribute functions, but using WLAN. In each case we have performed the cycle of operations starting with a fully charged battery and continuing until the device was not able to utilize the selected wireless interface any longer. Finally we compare both approaches to context distribution in terms of battery power consumption. We observe that Bluetooth consumes 2-6 times more energy for transmission of a 1MB file to two devices than to discover these two devices. Furthermore, the transfer of this file is two times slower than WLAN, and we must unicast this file to each device. Multicasting via WLAN proved to be *less energy consuming* than the Bluetooth transmission, if data is to be sent to more than three users. In addition, the energy to discover 2 devices along with their services using Bluetooth consumed 52 times more energy than to receive the same amount of data via a WLAN multicast. Thus, this paper shows that it is *more energy efficient to distribute context knowledge* to other devices, than having each device learn this information itself. Finally, we give equations for calculating the battery power consumption of transmitting data using any protocol that runs over Bluetooth or over WLAN.

MobiQuitous 2008, July 21–25, 2008, Dublin, Ireland.

Categories and Subject Descriptors
C.2.1 [**Network Architecture and Design**]: Wireless communication; D.4.8 [**Performance**]: Measurements; C.4 [**Performance of systems**]: Measurement techniques

General Terms
Measurement, Performance, Design, Experimentation.

Keywords
Battery power evaluation, context distribution, Bluetooth, WLAN

1. INTRODUCTION
In a context-aware system, devices are frequently mobile and geographically distributed, thus devices need to timely discover, collect, and adapt based upon context information from its surroundings. Here, context is used to describe the situation of an entity [1]. Alternatively, a device can share its context knowledge (which it has discovered and acquired) with other geographically distant devices (which have done the same) in order to learn about potential new contexts, *in advance* of arriving at a new location. Advance knowledge of context is powerful, because it can potentially reduce the delay or energy required by a device that needs to adapt to a new environment. If this context information is distributed in advance, then the query can be answered locally. There is a cost related to distribution of context that will never be used by another peer (in terms of communications, storage, and battery power consumption), but as much of this information changes slowly and this information can include other context, such the available projectors, scanners, printers, access points, power outlets, etc. in the same environment, the probability that none of this information is used decreases. Thus there will be a trade-off between how far context information should propagate and how useful this information is in advance (for adaptation by both the device and the user).

In order to understand this trade-off between the distribution of context data over a set of devices and the costs of this distribution versus its time-dependent value – we examined the battery power consumed by context discovery vs. context distribution.

Due to the limited power available, hardware sensors such as mobile phones, sport sensors, medical sensors, wireless keyboards, mice, etc. are typically equipped with short range radios, quite often Bluetooth class 2 or 3 radios. While more powerful devices, such as handheld devices, tablet PCs, laptops, etc. are equipped with both Bluetooth and WLAN network interfaces. As both technologies enable ad-hoc discovery & networking between heterogeneous devices, we evaluated the use of these technologies for context distribution within a local area (in this paper we consider a single hop).

The context distribution methods that we describe in this paper are based on a simple idea: each device discovers other nearby devices, collects context information from these discovered devices, and distributes this information to all the discovered devices, such that they all share the same (most recent) context information. We have performed these operations in cycles and measured the battery power consumption starting with a fully charged battery and continuing until the remaining battery power is too low to continue and the application is shut down by the operating system.

In this paper we present our preliminary results and experiences from performing these measurements over Bluetooth and WLAN on two different handheld devices: HP iPAQ 4150 and 6915. In order to evaluate the cost of sending small vs. big chunks of data, we append the context information each time it is collected to a file, which is in turn transmitted to all the discovered devices. As the file grows in size, we are able to collect data for different file sizes; hence we can use this data to estimate the amount of battery power required for any specific file size. Fitting this data to a formula for the cost of transferring data in terms of J/bit enables us to estimate the power consumption for any given protocol that will run over Bluetooth or WLAN. Therefore we can calculate the optimum frequency (with respect to the battery power consumption) to distribute and retrieve relevant context information.

This paper is organized in seven sections. Section 2 presents our proposals for context distribution using Bluetooth and WLAN. Section 3 explains the hardware and software used for measuring battery power consumption. Sections 4 and 5 describe how measurements were performed; discuss the results obtained using two different handheld devices, compare both wireless link technologies, and describe the insights gained by performing these measurements. Section 6 provides a brief overview of the related work. We conclude in section 7 with a recapitulation of the results and future plans.

2. CONTEXT DISTRIBUTION

2.1 Bluetooth Context Distribution

Context distribution using Bluetooth, as illustrated in Figure 1, works as follows: a device with a Bluetooth interface and a context distribution application initiates discovery of nearby devices along with the context information which each can provide and adds this information to a file, then sends this file to all discovered devices – in order to propagate this information.

As context information we are specifically interested in the list of services provided by a device. By propagating the complete list of all the discovered services, we can quickly generate a list of all services that all devices which are currently or soon could be in range have available. It is the distribution of the *aggregated information* which enables the discovery of devices and services *beyond the single hop limit*. Note that Bluetooth limits this distribution of context to seven or fewer simultaneous devices that are within a range of ten meters.

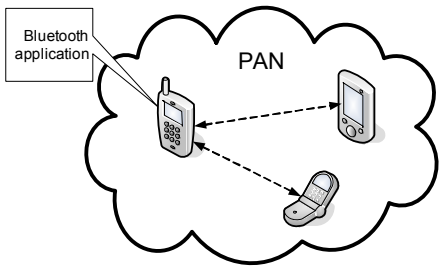

Figure 1. Bluetooth context distribution

Bluetooth service discovery can be done in two ways: by searching for a particular service or by browsing for available services using Bluetooth's Service Discovery Protocol (SDP) [2]. We utilize the latter (i.e., the browsing method), in order to learn about *all* available services offered by discovered devices [3].

File transfer is performed using Bluetooth's file transfer profile [4]. This file transfer profile depends on several underlying profiles and protocols. Two profiles handle discoverability and connection establishment. To transfer files the object exchange (OBEX) protocol is used. OBEX [4] allows a client to initiate a file transfer operation and push, pull, browse, or manipulate objects (files) on the server. The server needs to be available to other devices, accept incoming connections, and allow basic file transfer operations. The OBEX PUT operation is used to transfer objects to the server.

2.2 WLAN Context Distribution

In WLAN context distribution we assume that each device with a WLAN interface has already discovered some context and stored it in a local file. Distribution, as shown in Figure 2, works as follows: a device first enters a listening state, where it starts a timer with a random timeout value (initially selected between 6 to 9 seconds and increasing by 3 seconds each time the size of the merged file increases by an additional 150kB). Note that we have chosen these initial timeout values after experimenting with the protocol. This time is long enough to allow a server device to receive files from two other clients, but short enough to keep all devices synchronized for the entire measurement period.

When the timeout occurs, the device will check if it has received a *discoverPeers* message. If this message has been received, then the device acts as a client, sending a *peerReply* message followed by the current file containing the discovered context information. After sending the reply and file, the client will listen for a multicast of a merged file to arrive from the server. After receiving the merged file, the client returns to listening (in the *discoverPeers* state).

If the device determines after the timeout, that it has not received a *discoverPeers* message, then it will itself multicast this message, thus acting as a server. After multicasting this message, the server starts a timer and waits for *peerReply* messages and files from clients. Note that the client sends *peerReply* message

prior to the file transfer, thus there are two separate receive operations on the server side. When the timer expires, the server checks if *peerReply* messages and files have arrived and if so, it merges the received files into its existing file and multicasts the resulting merged file to all clients. Otherwise, it will multicast the existing file (generated in the previous round). After multicasting the file, the server returns to the *discoverPeers* state. For the next round, a new server will be randomly selected. In this way, context knowledge is shared among devices which are connected to the same wireless access point. However, this protocol could also be used on ad hoc WLAN networks.

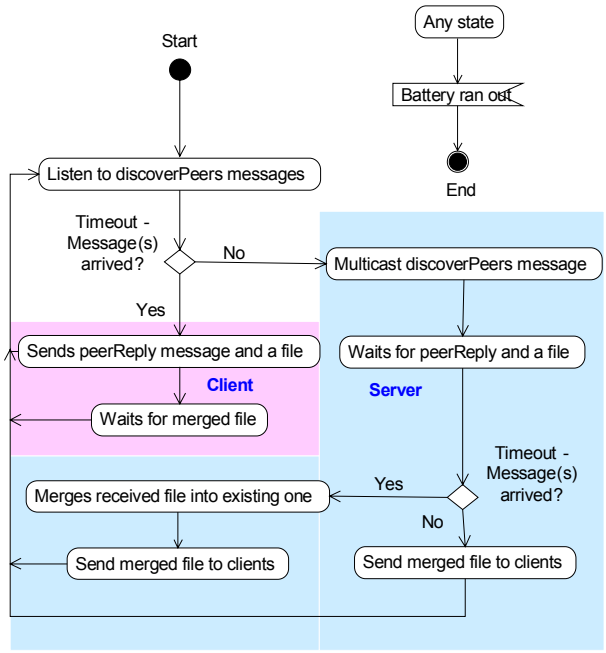

Figure 2. WLAN service discovery and file transfer protocol

3. MEASURING BATTERY POWER CONSUMPTION

3.1 Bluetooth and WLAN

Bluetooth in mobile devices generally uses one of two stacks: Widcomm/Broadcom stack [6] or Microsoft stack [7]. These APIs provide functions such as: device discovery, service discovery, and transfer of files between devices. We use High Point Software's BtAccess library [8], as the devices chosen both use the Widcomm Bluetooth stack. On top of this library we implemented our own context distribution application. To implement our WLAN context distribution we have used the System.Net.Sockets package (a part of the .NET Compact Framework).

3.2 Retrieval of battery power status

Most of today's mobile devices utilize a Smart Battery System [5]. Using Microsoft's coredll.dll library we developed for Microsoft Windows Mobile devices a P/Invoke function to retrieve the device's battery power status. The class SYSTEM_POWER_STATUS_EX2 is passed to the function as the first parameter.

To measure battery power consumption, we require the battery current and voltage values (as specified in this class). We also acquire the (remaining) battery life percentage values to provide information about the amount of battery life remaining as a percentage of the battery's initial full life time (capacity). We have developed a C# application that uses this library to log these values to a file.

3.3 Hardware

Measurements have been performed on two different types of devices (each of which has a separate internal backup battery to maintain data integrity during main battery replacement; additionally each has an integrated 802.11b WLAN interface). Table 1 shows characteristics of these devices. These devices were chosen because of their availability and differences in processing power, versions of the Bluetooth stack, and the battery capacity. Also both have the same type of Bluetooth stack and an integrated WLAN interface.

Table 1. Types of devices used in measurements

Device	Battery	CPU	Bluetooth stack
iPAQ 4150	Lithium ion, 1000mAh, 3.7V	400MHz Intel PXA255	Widcomm Bluetooth v1.4
iPAQ 6915	Lithium ion, 1200mAh, 3.7V	416MHz Intel PXA270	Widcomm Bluetooth v1.7

4. BLUETOOTH MEASUREMENTS

4.1 Measurements Description

During our measurements we have used 3 devices, each equipped with a Bluetooth interface. One device initiates device and service discovery, appends results to a file, and distributes this file to the other two discovered devices (each one acts as a remote device), responding to discovery inquiries, and retrieving the file. An application was developed and deployed that continuously performs and repeats Bluetooth device discovery, service discovery, and file transfer, as well as logging the battery current, voltage, and the remaining battery life (percentage) to a file for each of the activities (i.e., device discovery, service discovery, and file transmission). All phases were time stamped with both a start time and an end time.

In order to determine the battery power consumed for each operation, we have subtracted battery power values obtained in a measurement when the Bluetooth radio was turned off from the battery power values of each particular phase (see equation (1)). These measurements (with the Bluetooth interface turned off) were separately performed on each device. The values subtracted were chosen to match (in time) the battery power values in each phase of the series of Bluetooth operations. Note that device discovery is denoted as DD, service discovery as SD, and file transfer as FT.

$$P_X = \sum_{i=1}^{n} \left(U_{X_i} * I_{X_i} - U_{offX_i} * I_{offX_i} \right), \; X = \{DD, SD, FT\} \quad (1)$$

The overall measurement sequence is illustrated in Figure 3. The application was launched when the device was fully charged and continued until the battery level was too low to continue.

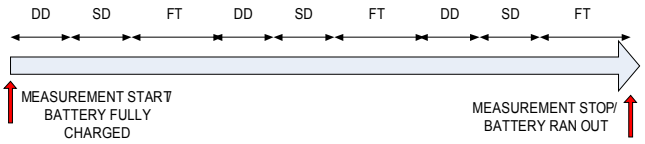

Figure 3. Measurement sequence

We append new discovery data and information about the battery power consumption to the file in each round. Therefore, the file size increases following each file transfer round. The information in the file is encoded in XML. In order to avoid reading and parsing the file when we need to append new information, the application simply seeks to the end of the file minus the length of the string of the last end tag, appends new data, and writes an end tag. This operation is constant in time and it takes less than 1ms to append new data to the file. As we could not measure this very brief operation using our application; we do not separate out the power required for this operation.

In our test environment we have three isolated devices: one master and two slaves. Their names were hard-coded in the application's discovery source code. They were initialized and configured to allow device discovery, service discovery, and file transfer operations initiated by the master device. The two different master devices used to perform measurements utilize two different versions of the Widcomm Bluetooth stack (as noted in Table 1).

4.2 Measurements Results

4.2.1 HP iPAQ 4150 Device

The measurements lasted for 8 hours, 6 minutes, and 14 seconds. While measurements performed on the same device in an idle state with the Bluetooth interface turned **off** lasted for 12 hours, 27 minutes, and 30 seconds, whereas when the Bluetooth was activated, but the device was idle, they took 11 hours, 26 minutes, and 32 seconds.

Figure 4 shows the battery consumption as a function of time calculated as P(t)=U(t)*I(t) from the values obtained from the measurements, where U(t) and I(t) represent the battery voltage and current values. The upper curve shows the battery consumption for all the activities performed during the measurements: discovery of two devices, discovery of services on a single device, and file transfer to a single device. It can be seen that battery power consumption (rate) is roughly constant. The lower curve shows the "corrected" battery power consumption, which is actually the battery power required during each phase of the measurement process reduced by the battery power values aligned in time when the Bluetooth interface was turned off and when the device was idle. This "corrected" battery power consumption for each Bluetooth phase is specified by equation (1). The average battery power consumed in the measurements was 335mW before and 109mW after subtracting the power when the Bluetooth interface was off.

Since the phases were not equally long (i.e., they took different amounts of time), one can not directly compare their battery power consumption; instead, we multiplied the average battery power (i.e., $\overline{P_x}$) consumed in each round with the average duration of each operation (i.e., $\overline{T_x}$) to obtain the average energy consumed from the battery (i.e., $\overline{E_x}$) due to each operation.

$$\overline{E_X} = \overline{P_X} * \overline{T_X} = \frac{\sum_{i=1}^{n}\left[\left(U_{X_i} * I_{X_i} - U_{offX_i} * I_{offX_i}\right) * T_{X_i}\right]}{n}, \quad (2)$$

$$X = \{DD, SD, FT\}$$

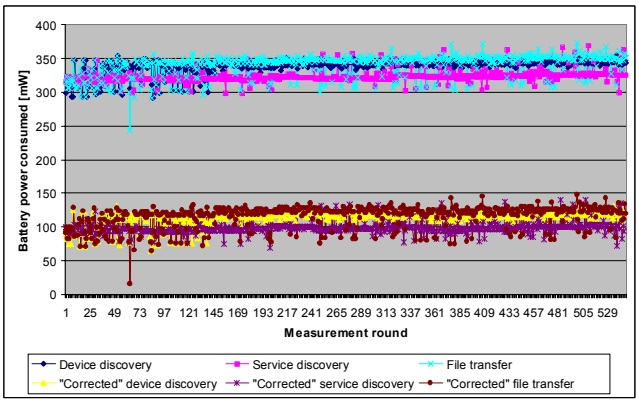

Figure 4. Comparison of battery power consumption for all three activities before (above) and after subtracting the power consumed when the BT interface was off (below).

Details about battery power consumption per activity are summarized in Table 2.

Table 2. Battery power consumed by all Bluetooth activities with a HP IPAQ 4150

Average	Device discovery	Service discovery	File transfer
power consumed: before correction	**339.7mW**	**324mW**	**340.3mW**
after correction	**114.6mW**	**98.6mW**	**114.9mW**
duration	**10.3 sec**	**1.6 sec**	**19.9 sec**
energy consumed	**1.18J**	**0.16J**	**2.35J**

Note that the size of the file that was transferred was 2.7kB at the beginning and 1.29MB at the end of the measurement period. The file size increases linearly in time, filesize(t)=2.34t +0.24, where the file size is expressed in kB and time in seconds.

Based upon the power consumed for the file transmission and the file transfer data rate we can estimate how many joules are consumed per transferred user data bit. The result is 3.9J/MB (i.e. 481.7nJ/bit) as obtained from the following equation:

$$Energy\_per\_transferred\_user\_data\_bit[J/bit] = \frac{P_{FT}[W]}{file\_transfer\_data\_rate[bit/s]} = \sum_{i=1}^{n}\left(U_{FT_i} * I_{FT_i} - U_{offFT_i} * I_{offFT_i}\right) \Big/ \sum_{i=1}^{n}\left(file\_size_i / T_{FT_i}\right) \quad (3)$$

Comparing the cost of device discovery (i.e., 1.18J) with the cost to transfer a 1 MB file (i.e., 3.9J) we can observe that the device consumes three times less energy to discover two devices than to transfer a 1 MB file to a single device. This is an important

result, showing that Bluetooth file transfer is not an energy efficient method to transfer data (as compared to WLAN). However, it is well suited for discovery of nearby devices.

Figure 5 shows the file transfer rate vs. file size. A logarithmic increase of the file transfer rate with the file size can be observed. Thus the file transfer rate initially increases with increasing file size up to a certain point; after which the file size does not significantly influence the file transfer rate. The maximum OBEX packet length is 255 bytes; therefore the file transfer always requires multiple OBEX packets. This means that more than one PUT request needs to be sent to and acknowledged by the server. The last PUT request will have the final bits set, thus indicating to the server that client is finished sending packets. There are no timeouts between OBEX packets.

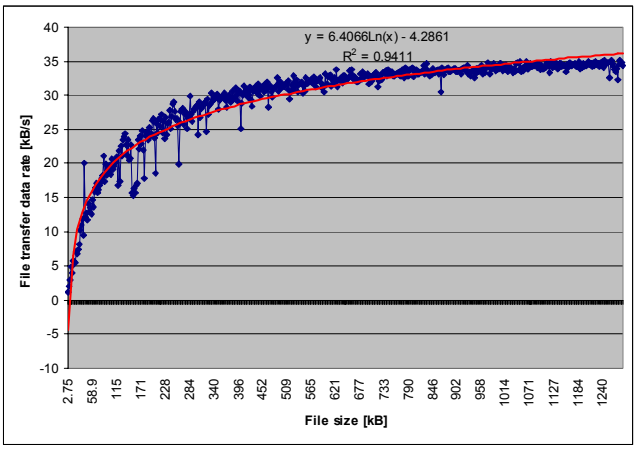

Figure 5. File transfer rate vs. file size

Using a maximum transmission unit of 255 bytes, the data rate at a file size of 520.3kB was 31.5kB/s and 33.9kB/s at a file size of 1MB. This MTU is too small to efficiently send large files, because sending more packets means waiting for more confirmation packets, which limits the transfer rate. We assume that the propagation time of responses is significantly less than the time required for processing packets of the file on the server side.

HP iPAQ 4150 uses a BRF6100 [9] Bluetooth single-chip which integrates Bluetooth baseband, RF, memory (ROM and RAM), and power management to enhance performance, reduce cost, and minimize board space. In its lowest power mode this Bluetooth transceiver requires 25mA in transmit mode and 37mA in receive mode at a supply voltage of 1.8V. Based upon this we can calculate the power, as 45mW in transmit mode and 66.6mW in receive mode. According to [10] the energy required to transmit a single burst of data from an initially powered-down transmitter can be expressed as follows:

$$E_{tx}(N, R_C, P_{amp}) = P_{start} * T_{start} + \frac{N}{R_C * R}(P_{txElec} + P_{amp}) \quad (4)$$

The two terms in the expression represent the energies for startup and transmission, respectively. Where P_{start} and T_{start} represent the power and latency of radio startup, P_{txElec} is the active transmission power, P_{amp} the dissipated amplifier power, N the number of bits before FEC, R the radio bit rate, and R_C the convolutional rate. Assuming that the energy consumed for the startup was significantly lower than the energy consumed for transmission (as the time needed for transmitter startup is in the order of hundreds of milliseconds versus seconds of transmission time and $P_{start} << P_{txElec}$), we assume values for the lowest power consumption where $P_{amp} \approx 0$dBm, $R_C=0.5$, and use a radio transmit data rate as R=721kbps, this results in an energy consumption per transmitted bit of:

$$\frac{E_{tx}}{N(1bit)} \approx \frac{P_{txElec}}{R_C * R} \approx 1.25 * 10^{-7} J/bit = 125 nJ/bit \quad (5)$$

When comparing this result with the value calculated from our measurements using (3) (i.e., 481.7nJ), we can see that they are of the same magnitude; with this calculated value being smaller, since this computation assumed the *lowest* power consumption case. Because our measured value includes all the other operations required to actually get the bits to transfer in addition to effectively transferring the user data bits we expect it to be higher. It is important to note that we are computing in equation (5) the energy consumed to send a single bit from a transmitter, while the earlier calculations concerned the total energy to send a single user data bit, not including the coding of the user bits (or the header bits) nor the extra overhead bits which are set for the lower layer protocol (for example framing, addressing, synchronization, polling/response, link layer management, etc.), or the time (and energy) listening to be polled, waiting for an acknowledgement, and the protocol overhead at the higher layers. We note that the ratio between the energy consumed per transferred user data bit and the energy consumed to transfer a single bit from the transmitter is 3.9.

4.2.2 HP iPAQ 6915 Device

For this device the measurements lasted for 9 hours, 2 minutes, and 49 seconds until the remaining battery power was 12% of the full battery capacity[1]. If the measurement would have run until the battery was at zero capacity, the estimated duration of these measurements would be 10 hours, 23 minutes, and 22 seconds. For comparison, the measurements on the same device in an idle state with Bluetooth interface turned off lasted for 23 hours, 1 minute, and 14 seconds, whereas when the Bluetooth was activated, but the device was idle, they took 19 hours, 23 minutes, and 4 seconds.

Figure 6 shows the battery consumption for all activities performed during the measurements. The lower curve shows the battery power consumption after correction (in the same way as described in §4.2). The average battery power consumed during the measurement was 405.5mW before and 231.9mW after correction. The lower curve also shows several power values close to zero - these are errors due to the simple way in which the measurement values were corrected.

[1] The BT device in the iPAQ 6915 could no longer be used once the battery voltage dropped below 3.664V. At this time we are not certain why this is true, but suspect that the OS in purposely turning off the device to save some remaining battery power. Note that in our earlier measurements with HP iPAQ 5550 the operating system turned off the WLAN interface at some point to preserve some operating time without the WLAN.

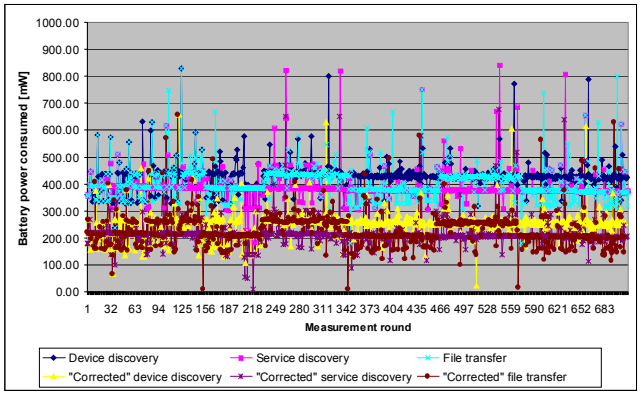

Figure 6. Comparison of battery power consumption for all three activities before and after correction

Details about battery power consumption per activity are summarized in Table 3.

Table 3. Battery power consumed by all Bluetooth activities for a HP iPAQ 6915

Average	Device discovery	Service discovery	File transfer
power consumed: before reduction	**425.7mW**	**388.4mW**	**402.5mW**
after reduction	**251mW**	**215.3mW**	**229.3mW**
duration	**11.7 sec**	**2.1 sec**	**10.9 sec**
energy consumed	**2.93J**	**0.43J**	**2.49J**

Applying the same equations used earlier for the HP iPAQ 4150, we calculate that the energy consumed (until the point when the remaining battery power was 12%) for each Bluetooth phase. As shown in Table 4, these results are 2-3 times higher for the device and service discovery, and almost the same for file transfer. Table 4 summarizes the energy consumption results for both devices: average energy consumed for device (DD) and service discovery (SD), energy consumed per transferred user data bit (FT), as well as the total energy consumed for all rounds.

Table 4. Comparison of energy consumptions

HP iPAQ	DD	SD	FT (per user data bit)	Total energy consumed
4150	1.18J	0.16J	481.7nJ/bit	3383.4J
6915	2.93J	0.43J	401.5nJ/bit	6203.5J (88%) 7334.7J (estimated for 100%)

HP iPAQ 6915 uses a BRF6150 [11] Bluetooth single-chip solution. In its lowest power mode, its Bluetooth transceiver utilizes 25mA in transmit mode and 37mA in receive mode at a supply voltage ranging from 1.8V to 3.6V. Calculating the power, we obtain 67.5mW (using the mean voltage value) in transmit mode and 99.9mW in receive mode. When calculating the energy consumed per transmitted bit, we get:

$$\frac{E_{tx}}{N(1bit)} \approx 1.872*10^{-7} J/bit = 187.2nJ/bit \qquad (6)$$

Figure 7 shows the file transfer rate vs. file size, which follows the same trend as in Figure 5, but with significantly higher data rates. It can be seen that the data rate of this device is 2.4 times faster than the rate of the other model. The size of the file that was transferred was 2.7kB at the beginning and 1.48MB at the end of the measurement period. The data rate at a file size 519kB was 75.95kB/s and 82kB/s at a file size of 1MB.

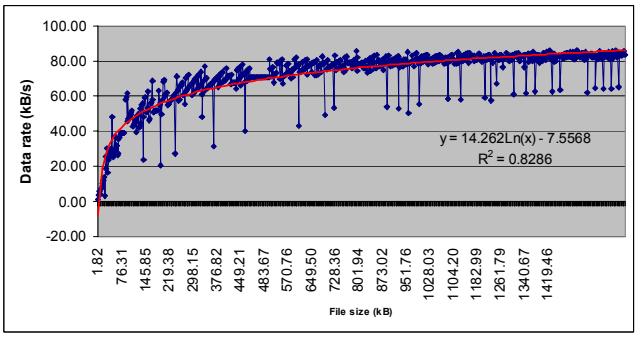

Figure 7. File transfer rate vs. file size

Moreover, the total energy consumed by the iPAQ 6915 for all rounds was 6203.5J. The estimated total energy consumed (given the ability to fully utilize the battery capacity) is estimated to be 7334.7J, while the total energy consumed by 4150 device was 3383J. Thus we observe that iPAQ 6915 consumed twice as much energy from the battery as the iPAQ 4150. Possible reasons for this are that iPAQ 6915 has a faster processor than iPAQ 4150 (Intel PXA270 at 416 Mhz vs. Intel PXA255 at 400Mhz), a newer Bluetooth stack (version 1.7 vs. 1.4), thus **waiting** for Bluetooth input consumes significantly more battery power because the processor is not being put into a low power mode, despite the fact that it is to perform the same operations as the iPAQ 4150. Another reason is that the set of measurements ran (11%) longer on the iPAQ 6915 than on the iPAQ 4150, thus we would already suspect that unless the iPAQ 6915 consumed less energy on average per operation than the iPAQ 4150 – that its total energy consumption would be greater – however it is more than proportionally greater.

Note that in order to estimate the total energy consumed for the full battery capacity, we first calculated the sum of the estimated durations of all phases for the rest of the (estimated) time (that the measurements would run if we were able to continue to operate until there was no battery power left), multiplied by the average battery power of each phase, and add them to the already calculated energies.

5. WLAN MEASUREMENTS

5.1 Measurements Description

In the WLAN measurement we have also used 3 devices, each equipped with a WLAN 802.11b interface, in conjunction with a D-Link DI-524 high speed IEEE 802.11g wireless router, which is 802.11b compatible. As explained in §2.1, after listening for a random period, a device which times out attempts to assume the

role of being a server, while the other (two) devices act as clients.

A randomly selected measurement sequence is shown in Figure 8. The client's and server's activities are also illustrated as a function of time. First we measure the duration of each activity which a device performs, and determine the corresponding battery power consumed for this operation, these results can be correlated and compared with the results from the earlier Bluetooth measurements.

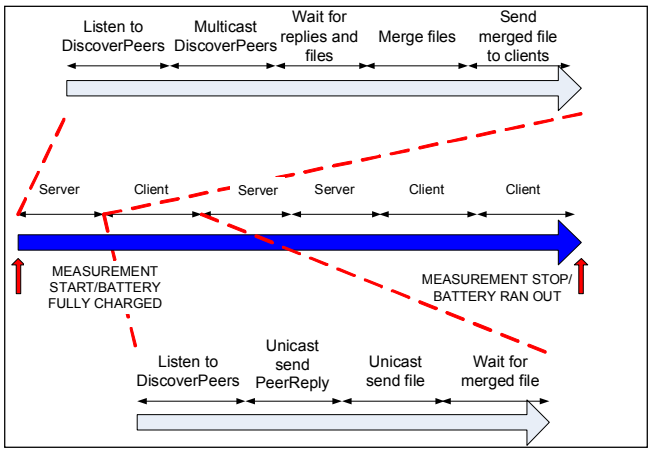

Figure 8. WLAN measurement sequence

In order to determine the battery power consumed for each of the client and server operations, we have subtracted the battery power values obtained when the WLAN interface was turned off. These measurements (with the WLAN interface turned off) were performed separately on the same device and the values were used to correct the power consumed in each of the phases of the WLAN operations.

Bluetooth generated files that contain context information on each client were sent to a server, which merges them into a single file and sends this file back to other clients. A merged file is composed at the beginning of the measurement period and the battery information for each activity a device performs is appended in each round to this file. The reason for doing so is to simulate the increasing file size in order to be comparable to the earlier Bluetooth measurement, while avoiding the exponential increase in file size caused by *appending* a newly generated merged file to the existing one.

5.2 Measurements Results

5.2.1 HP iPAQ 4150

The measurements lasted until the device could no longer operate the WLAN interface – which took 2 hours, 14 minutes, and 42 seconds, i.e. four times shorter than the measurement period of the Bluetooth measurements. After analyzing this data, we eliminate the power consumed in following phases (because they were too short to be considered in the energy consumption calculation): the server's multicast of the *DiscoverPeers* message, the client's sending of the *PeerReply* message, and the client's sending of the Bluetooth generated file.

Figure 9 shows the battery power consumption for activities that a device performs in the server role: listening to *DiscoverPeers* message, waiting to receive peer replies & files from clients, and sending a merged file to discovered clients. Note that the first operation relates to the blocking receiving function on a UDP multicast socket, while the other two activities are receiving and sending a file over a TCP socket, respectively.

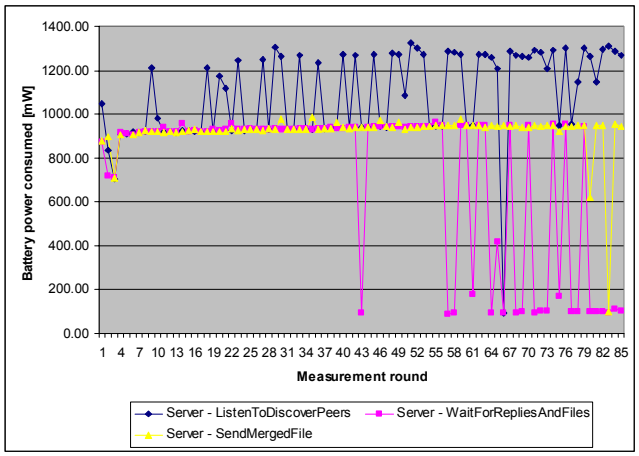

Figure 9. Comparison of battery power consumption for all server activities after subtracting the power consumed when the WLAN interface was off

It can be seen that listening to *DiscoverPeers* is the most power consuming operation, which varies between 912mW and 1325mW. Note that a device in this phase does not receive any *DiscoverPeers* message after the timeout expires, because it acts as a server and multicasts this message. Note also that the device had the automatic power save mode[2] turned on during the measurement period, meaning that the WLAN card enters a *Sleep* state [12] (where a majority of the circuitry is switched off, except for some critical parts) after a certain elapsed period of inactivity. It wakes up after a preset interval to check for the traffic queued for it at the access point. The battery power consumption of a WLAN device in the idle (after it was reduced by the values when WLAN was off) is shown in Figure 10.

Since listening for *DiscoverPeers* activity is realized by the blocking socket receiving function, we assume that the majority of the time the device is in the *Listen* state [12] and that it does not go to the *Sleep* state. In the *Listen* state a device listens for the (multicast) traffic, but does not pass any data to the host. We also assume that instantaneous power consumption, illustrated by periodic peaks, corresponds to short periods when the device was receiving an announcement frame from the access point. The announcement period (DTIM (Delivery Traffic Indication Message) interval) was set to 300 ms.

The waiting for *PeerReplies* & files and sending a merged file have very similar power consumptions in the first half of the measurement period. The reason for this similarity is that they both receive and send data via a TCP socket. The difference is that the waiting for *PeerReplies* and files operation is terminated after the preset timeout value, which increases with the merged file size increase (as explained in §2.2), while the sending of a

[2] The automatic power save mode is by default set by the device manufacturer because it achieves the maximum power saving without degradation of performance.

merged file operation finishes immediately after the data transmission completes. Note that the receive operation is non–blocking, meaning that if there is no data to receive, the device will be idle, and will go to the *Sleep* state. This could explain the occurrence of the instantaneous drops of the battery power to the same low values as when the WLAN was idle (see Figure 10). A drop in battery power consumption also happened during the send merged file operation just before the end of the measurement period. This can be seen in Figure 11, and a possible explanation is that the device briefly disconnected from the access point.

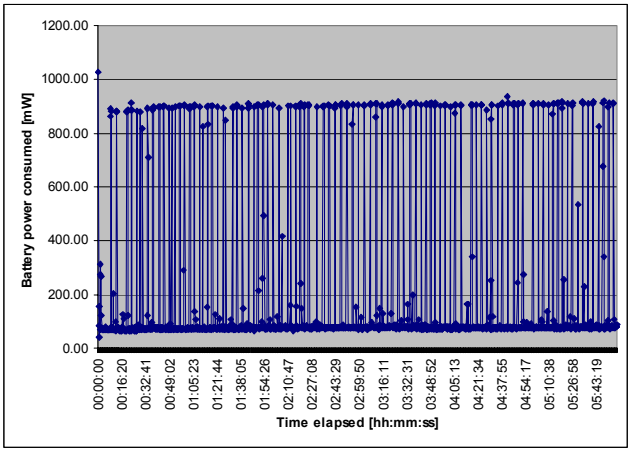

Figure 10. Battery power consumption plot for WLAN idle measurement after correction

Figure 11 shows the battery power consumption for activities that a device performs as a client: listening for the *DiscoverPeers* message and receiving a merged file from the server. In the listening for *DiscoverPeers*, the client actually gets the message, but this retrieval is too short to be captured in the log files because the whole message fits into a single packet of 1024 bytes. Thus, its battery power consumption is the same as in the server role.

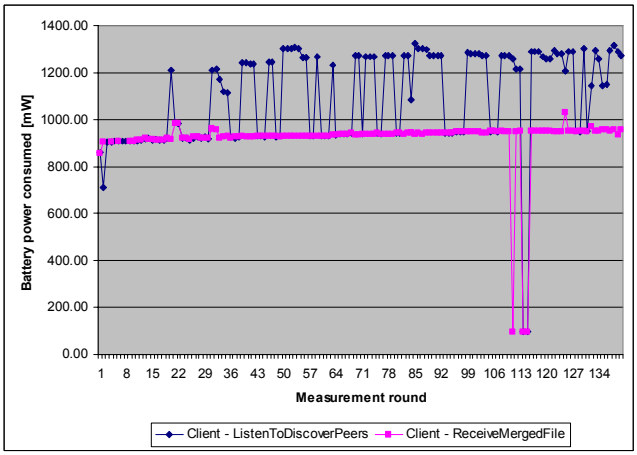

Figure 11. Comparison of battery power consumption for all client activities after correction

It can be noticed that the power consumption of receiving the merged file is the same as the sending of the merged file. Three decreases in battery power consumption can be observed in Fig.11, which can be explained as the client device not receiving the file because of a temporary disconnection from the access point.

Tables 5 and 6 summarize the details of the average power and energy consumption per activity in the server and client role. We can observe that the most energy was consumed during blocking receive operations when listening for a *DiscoverPeers* message, while the least energy was spent to actually send and receive data. In waiting for *PeerReplies* and files operation the device consumed less energy than in the listening activity, due to the non-blocking receiving operation.

Table 5. Battery power consumed by WLAN server activities for a HP iPAQ 4150

Average	Listen to *DiscoverPeers*	Wait for *PeerReplies* and files	Send merged file
power consumed: before reduction	**1312.7mW**	**959.3mW**	**1151.8mW**
after reduction	**1083.4mW**	**728.4mW**	**920.7mW**
duration	**15sec**	**15sec**	**13sec**
energy consumed	**16.67J**	**9.96J**	**11.5J**

Table 6 Battery power consumed by WLAN client activities for a HP iPAQ 4150

Average	Listen to *DiscoverPeers*	Receive a merged file from a server
power consumed: before reduction	**1316.5mW**	**1144.5mW**
after reduction	**1089.8mW**	**919.2mW**
Duration	**14.9 sec**	**4.5 sec**
energy consumed	**16.6J**	**4.1J**

To compare the cost of WLAN to discover two devices with the Bluetooth device discovery, we needed to compare the energy consumptions of the corresponding Bluetooth and WLAN activities. Bluetooth device discovery corresponds to the WLAN's listen for *DiscoverPeers* and receiving *PeerReplies*. However, sending and receiving of a *PeerReply* message takes a very short time and can be ignored in the energy consumption calculation. Thus, we can conclude that Bluetooth device discovery consumed significantly less energy than its WLAN counterpart (1.18J vs. 16.6J). Looking at their average durations, it also took less time to discover two devices via Bluetooth (10.3s) than in WLAN (15s). However, one should note that the duration of the listen for *DiscoverPeers* phase was set programmatically by the random timeout value that increases with the merged file size (see §2.2).

Figure 12 shows the transfer rate vs. file size of multicasting this file to two devices and the results of fitting this to a four degree polynomial function. The size of the merged file was 16.8kB at the beginning and 751.7kB at the end of the measurement period. The data rate at a file size 520kB (per device) was 57.8kB/s and 57.2kB/s at a file size of 743.6kB. These values are lower than expected, since sending of the merged file was performed by writing data to the socket while reading from the file in parallel.

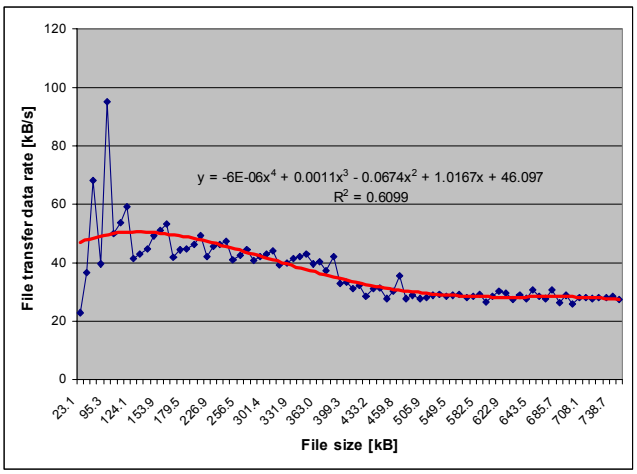

Figure 12. File transfer rate vs. file size

After calculating the energy consumed to transfer a single user data bit using our formula, we got 1.56μJ/bit, meaning that 3.2 times more energy was consumed per bit by sending data over WLAN than over Bluetooth. Additionally, WLAN is 1.8 times faster than the Bluetooth (2*16s in Bluetooth and 18s in WLAN to transfer a 500kB file to two devices). Because we can send data over multicast to multiple users at once, this result tells us that distributing data over WLAN is more power efficient method than using Bluetooth when the number of recipients exceeds three.

We showed by now that context discovery should be done by Bluetooth and context distribution using the WLAN multicast. In order to answer the question if it is better to perform context discovery or context distribution, we will compute how many joules are consumed by a client to receive a single user bit over the WLAN multicast:

$$Energy\_per\_received\_user\_data\_bit[J/bit] = \frac{E_{RMF}[J]}{received\_file\_size[bit]} \quad (7)$$

$$= \sum_{i=1}^{n} \left(U_{RMF_i} * I_{RMF_i} - U_{offRMF_i} * I_{offRMF_i}\right) * T_{RMF_i} \Big/ \sum_{i=1}^{n} received\_file\_size_i = 1.33\mu J/bit$$

Note that E_{RMF} in the equation (7) is the total energy consumed by a device to receive a merged file. Comparing this result of 1.33μJ/bit with the average energy consumed by Bluetooth to discover two devices along with their services (i.e., 1.5J), we can observe that a device would spend significantly less energy to discover 2 other devices and their services (approx. 2.7kB of data) then to receive a file of the same size over WLAN multicast (i.e., 28.7mJ). Note that to consume 1.5J, a device could receive the file of 140kB over WLAN multicast. Moreover, the energy to discover context would increase with the number of nearby devices. Therefore, it is **more energy efficient to distribute** (once discovered) **context information** to other devices in advance (using WLAN multicast), rather than having all devices learn this information themselves.

6. RELATED WORK

In the literature there are many context-aware frameworks for enabling mobile devices to adapt their configuration to environmental conditions. All of these frameworks need to employ some context discovery and distribution mechanism in order to provide the right context anywhere, anytime. Some of the most popular frameworks are the Context Toolkit [16], the Service-Oriented Context-Aware Middleware (SOCAM) [17], and the Context Broker Architecture (CoBrA) [18]. However, all of them are based on a centralized discovery mechanism where distributed entities that provide context information (i.e. sensors, context providers) have to register in order to be discovered. A pure peer-to-peer context-aware system such as Hydrogen [19] uses the device's local context, i.e. context acquired by local built-in sensors. Due to its limited capabilities a device cannot sense all the context information itself, Hydrogen provides a mechanism to share sensed context with other nearby devices. Context sharing is based on a peer-to-peer connection over LAN, WLAN, or Bluetooth. However, authors do not mention distributing the "aggregated context", i.e., context originating from two or more devices, which can be exchanged with a newly encountered device in order to learn about context beyond a single hop. In [20], authors designed a ubiquitous-oriented peer-to-peer context sharing model (PCSM) that constructs channels for remote registration of Context Database Agents through a Registration Query. Although this model is well designed for disconnected operations by using lightweight messages, the authors did not investigate the communication and battery power costs of exchanging small vs. big chunks of context data.

A lot of research has studied battery power measurements for mobile devices, in particular for optimization. However, only a small amount of work targeted context. In [14], the authors propose a system for enabling applications running on mobile devices to adapt their behavior in order to reduce their energy consumption, by optimizing the collaboration between applications and the underlying operating system. A similar goal drives the research illustrated in [13], where the authors propose an energy-aware QoS model (e-QoS) providing QoS guarantee in terms of energy consumption of network-centric applications running on mobile devices. This is accomplished by dynamically selecting and adapting application protocols. Finally, the work described in [15] introduces a system for context aware battery management, based on prediction algorithms, which helps a mobile device user to prevent a complete battery discharge.

7. CONCLUSION AND FUTURE WORK

In this paper, we introduced and evaluated context distribution methods in mobile systems environments using Bluetooth and WLAN technologies. The evaluation of the proposed methods was performed by collecting and comparing battery power consumption measurements. Such measurements were performed separately on two different models of Bluetooth and WLAN enabled mobile devices. We have seen that the HP iPAQ 6915 device consumed twice the energy of the HP iPAQ 4150 to perform the same Bluetooth operations (i.e., device discovery, service discovery, and file transfer). The possible reasons for this

are that iPAQ 6915 has a faster processor than iPAQ 4150, and a newer Bluetooth stack, all of which lead to more battery power being consumed to perform the same operations. We found out that Bluetooth consumes 2-6 times more energy to send a file of 1MB size to two devices than to discover them – hence **distributing** this information **via Bluetooth** is more expensive than directly **learning** it! Additionally, the file transfer by the model 6915 was 2.4 times faster, however we do not know whether this is due to the newer BT stack or the faster CPU.

The WLAN measurements were performed (at the time of writing) only on iPAQ 4150. The results showed that the energy consumed per transferred user data bit was 1.56μJ/bit for WLAN vs. 481.7nJ/bit for the Bluetooth file transfer (a ratio of 3.2). Additionally, the WLAN transfer is faster than the Bluetooth, taking a half of the time to transfer the same amount of data between two devices. Therefore, if data is sent to more than three devices at once via **WLAN multicast** this **is more energy efficient than** using **Bluetooth**.

By comparing the energy used to discover two devices and their services (i.e., 1.5J) with the energy that would be consumed to receive the file of the same size (i.e., 28.7mJ), we concluded that it is **more energy efficient to distribute** (once discovered) **context information** to other devices in advance, rather than having all devices need to learn this information themselves.

We also found out that the main reason that WLAN consumed more energy than the Bluetooth was a **long timer value** set programmatically on the WLAN **to discover devices**. The blocking receive operation in the WLAN discovery phase did not let the processor go into its low power mode. Therefore, we plan to shorten this timeout value in the WLAN protocol implementation and modify the measurement application to include the periods of a processor's activity and inactivity, in order to be able to estimate the difference between the energy consumption of a device performing vs. waiting for an operation.

Note that this paper did not explicitly address the issue of the time waiting for link layer acknowledgements. However, measuring the details of the effects of waiting would require additional experiments and might be subject for a future paper.

8. ACKNOWLEDGMENTS

The authors of this paper would like to thank their partners in the MUSIC-IST project and acknowledge the partial financial support given to this research by the European Union (6th Framework Programme, contract number 35166). We would also like to thank Prof. Gerald Q. Maguire Jr., for his fruitful discussions and valuable comments during this research work.

9. ADDITIONAL AUTHORS

Additional authors: Alessandro Mamelli (HP Innovation Center, Milan, Italy, e-mail: alessandro.mamelli@hp.com) and Athanasios Karapantelakis (Ericsson AB and Royal Institute of Technology (KTH), Stockholm, Sweden, e-mail: athkar@kth.se).

10. REFERENCES

[1] Dey, A., 2000., Providing Architectural Support for Building Context-Aware Applications, PhD thesis, College of Computing, Georgia Institute of Technology, AT, USA

[2] Specification of the Bluetooth system, Core Specifications, version 2.1 + EDR, v2.0 + EDR, v1.2, and v1.1

[3] Institute of Electrical and Electronics Engineers, 1999. Short Description of the Standard, IEEE P802.11 working group

[4] Bluetooth Special Interest Group, 2001. Specification of the Bluetooth System: Profiles, Volume 2, Version 1.1

[5] Benchmarq Microelectronics Inc. et al, Smart Battery Data Specification, Smart Battery System Specification report, rev. 1.1, 1998.

[6] Broadcom Inc., Widcomm Bluetooth software, http://www.broadcom.com/, last visited on 16-March-2008.

[7] Microsoft Inc., Bluetooth stack architecture, http://msdn2.microsoft.com/de-de/library/ms890956.aspx, last visited on 16-March-2008.

[8] High Point Software Inc., BtAccess home page, http://www.high-point.com/, last visited on 16-March-2008.

[9] Texas Instruments, BRF6100 Fully Integrated Bluetooth Transceiver, Product Brief, 2002.

[10] Rex, M. and Chandrakasan, A., A Framework for Energy-Scalable Communication in High-Density Wireless Networks, ACM ISLPED'02, Monterey, CA, USA, August 2002.

[11] Texas Instruments, BRF6150 Bluetooth Specification v1.2 single-chip solution, Product Brief, 2004.

[12] Atheros Comm., Power Consumption and Energy Efficiency Comparisons of WLAN products, White paper, 2003.

[13] Lufei, H. and Shi, W., Energy-Aware QoS for Application Sessions across Multiple Protocol Domains in Mobile Computing, Computer Networks: International Journal of Computer and Telecommunications Networking, Vol. 51, No. 11, 3125-3141, August 2007

[14] Flinn, J. and Satyanarayanan, M., Energy-aware adaptation for mobile applications. In Symposium on Operating Systems Principles, pp. 48-63, December 1999.

[15] Ravi, N., Scott, J., Han, L., and Iftode, L., Context-aware Battery Management for Mobile Phones. IEEE International Conference on Pervasive Computing and Communications (PerCom 2008), Hong Kong, China, March 2008.

[16] Dey, A.K. and Abowd, G.D., 2000. The Context Toolkit: Aiding the Development of Context-Aware Applications, In the Workshop on Software Engineering for Wearable and Pervasive Computing, Limerick, Ireland, June 6, 2000.

[17] Gu, T., Pung, H.K. and Zhang, D.Q. A service-oriented middleware for building context-aware mobile services, In Proc. of IEEE Vehicular Technology Conference (VTC), Milan, Italy, August 2004.

[18] Chen, H., An Intelligent Broker Architecture for Pervasive Context-Aware Systems, PhD Thesis, University of Maryland, Baltimore Count, 2004.

[19] Hofer, T., Schwinger, W., Pichler, M., Leonhartsberger, G., and Altmann, J., 2003. Context-Awareness on Mobile Devices – the Hydrogen Approach, In Proc. 36th Annual Hawaii International Conf. on System Sciences, pp.292-302, January 2003.

[20] Ye, J. Li, J., Zhu, Z., Gu, X., and Shi, H., PCSM: A Context Sharing Model in Peer-to-Peer Ubiquitous Computing Environment, In International Conf. on Convergence Information Technology, Gyeongbuk, Korea, IEEE Computer Society, pp. 1868-1873, November 2007.