University of Zurich

Enabling a mobile phone to sense its surroundings

Integration of internal and external sensors in a mobile phone

Diploma Thesis in Computer Science

submitted by Robin Bucciarelli
Zurich, Switzerland

student number 99-700-320

written at the Department of Informatics
University of Zurich
Prof. Abraham Bernstein, Ph.D.

supervised by Peter Vorburger

submitted on 05.01.2006

Abstract

In the era of wireless communication, so-called smartphones have inexorably pervaded our way of live. Through their ubiquity, the user is exposed to recurrent interruptions, oftentimes unwanted ones. Consequently, the question emerges whether and to what extent a smartphone truly could become smart in order to minimize these interruptions. To find an answer to this question, we enabled a smartphone with the possibility to ascertain its own surroundings, that is to become context-aware.

The goal of this thesis is the preparation of an experiment approaching the vision of a truly “smart” phone. This consists in the design, the implementation and the evaluation of an application capable of collecting and persistently storing data originating from sensors mounted in and around a smartphone.

Zusammenfassung

Im Zeitalter mobiler Kommunikation haben sogenannte Smartphones unaufhaltsam unsere Lebensweise durchdrungen. Durch ihre Allgegenwart ist der Benutzer wiederholten Störungen ausgesetzt. Folglich stellt sich die Frage, ob und inwiefern ein Smartphone tatsächlich in der Lage sein könnte, selbständig diese Störungen zu minimieren. Um darauf eine Antwort zu finden, wurde ein Smartphone in die Lage versetzt, ein eigenes Kontextbewusstsein zu entwickeln.

Das Ziel dieser Diplomarbeit ist ein Experiment, welches der Vision eines wirklich „smarten“ Smartphones einen Schritt näher kommt. Dies besteht aus dem Entwurf, der Implementierung und der Evaluation eines Programms, welches Daten, die ihrerseits von an und um das Smartphone angebrachten Sensoren stammen, sammeln und persistent speichern kann.

Contents

Chapter 1

Introduction

The way of life of many people has experienced a considerable acceleration since the invention of mobile phones. These devices have become an integral part of everyday life, accompanying us wherever we go. This ubiquity gives rise to a constant availability on our part, thus exposing us to interruptions[1]. Most of the time, such an interruption distracts us from an ongoing task and forces us to react to it. Due to the partially dangerous nature of certain interruptions, e.g. while driving a motor vehicle, it would be desirable to have a mobile phone able to decide whether to forward or block a call depending on the user's surroundings.

A typical scenario illustrating an unwanted interruption and the inconvenience caused to the user is described in the following paragraph:

Mr. Sellars is an avid movie fan. He often goes to the movie theatre to watch the newest motion-pictures. Normally he would turn off his mobile phone as soon as the movie starts playing. But since he's a doctor always on stand-by for possible emergency duties, the mobile phone has to stay activated. The majority of calls he receives during his visit to the movie theatre however don't originate from his workplace, causing nonetheless an unwanted distraction. In such situations Mr. Sellars wishes nothing more than to possess a truly smart smartphone which knows when and if to disturb its owner…some sort of artificial receptionist perhaps.

[1] Interruptions caused by mobile phones include incoming calls and all sorts of messages (SMS, MMS, etc.).

In order to ascertain a user's interruptability, one has to be aware of its context. A possible definition of the term context is given in [Dey 01]:

> *Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*

The human race has learned to intuitively sense one's context. In social interactions, for example, it is possible to detect the counterpart's intentions by means of facial expressions and gestures[2]. Context therefore is detectable by implicit and contiguous information.

A machine, in our case a smartphone, first needs to gather information about its surroundings in order to discern its context. To do that, it needs sensors capable of delivering potentially useful information about the ever-changing environment.

[2] Even in this case, the context can only be sensed if both dialogue partner originate from the same culture area.

1.1 Goal

The main focus of this thesis lays on the integration and the unification of all relevant data streams into a single application, to gather data concurrently and if possible, as a background process. This procedure constitutes the foundation to conduct the experiment, which in turn converges on the vision of a truly “smart” phone. A smartphone, namely the Treo 650 of Palm Inc.[3], acts as the carrier of the controlling software as well as the repository for the gathered data.

The data streams can be roughly divided into two categories: phone ***internal*** and ***external*** sensors.

Internal sensors comprise all information that the device itself is able to gather. Ranging from network information such as the signal strength, the roaming status and cell ID up to downright data streams like the audio recorder and the photo camera.

External sensors on the other hand rely on third party hardware. In our case they comprise a GPS device, connected via Bluetooth and a custom made sensorboard with an extensive array of sensors on it. The sensorboard is a proprietary development of Peter Vorburger at the Dynamic and Distributed Information Systems group, Department of Informatics, University of Zurich. It is composed of multiple sensors which will be more closely examined in the requirements section.

By integrating aforementioned data streams into the smartphone and storing them, in our case, on an expansion card, we offer the possibility to conduct data mining on them at a later date, which in turn will help predict how the smartphone reacts to interruptions such as incoming calls.

[3] Detailed information about the Treo 650 smartphone can be found in Appendix A.

1.2 Thesis outline

After the brief general introduction into context awareness and the motivation behind this thesis in **Chapter 1**, follows **Chapter 2** describing a possible future working scenario, from which an exhaustive listing of requirements is deduced.

Chapter 3 addresses the implementation part of the requirements with emphasis on the integration of the data sensors. After an excursion into general application design follows an in-depth presentation of the developed applications. A mention of constraints encountered during the implementation phase concludes this chapter.

Measurements regarding hardware components (battery life) and evaluation and presentation of the collected data constitute the subject matter of **Chapter 4**.

In the end, **Chapter 5** summarizes the obtained achievements, mentions potential hot spots where further work has to be done and gives an overview of future developments as well as the author’s personal assessment on the project as a whole.

Chapter 2

Requirements

- Overview
- Future scenario
- Key requirements
    - Unobtrusive application
    - Phone event interception
    - Context detection
        - Internal Sensors
        - External Sensors
    - Parallel processing
    - Data logging

This chapter addresses the requirements needed to develop and implement an application able to attain an understanding of its context by analysing various senor data thereupon deducing the interruptability of a person. To understand the intention behind these requirements, section 2.1 will briefly illustrate a vision of a possible future scenario based upon this technology, followed by a closer look at the various key requirements in section 2.2.

2.1 Future scenario

At first, we will examine how an ideal scenario looks like by means of a fictitious example illustrating the semi-autonomous behaviour of our application during incoming phone calls. The example subsumes the past and future work of several other diploma theses in order to give the reader a comprehensive overview of the experiment in its entirety.

Mr. Sellars is once again seated in his favourite movie theater watching a motion-picture. This time though his cellular phone is equipped with our "Artificial Receptionist[4]" application. During the movie the smartphone receives an incoming call, which promptly gets intercepted by our application. Due to the evaluation of the sensor data delivered by the device, the application successfully identifies the surroundings the user stays in and sets the device's profile to "do not disturb" as per user request. But since Mr. Sellars is a doctor working in a hospital always on stand-by for possible emergency duties, certain emergency numbers always overrule the current settings thus allowing the call to be forwarded. This time though the call originated from his mother-in-law's cell phone which got instantly blocked and rerouted to his answering machine for later consideration.

The above mentioned experiment demonstrates the operating mode of our application in collecting potentially useful information about a user's context such as his current activity, the surroundings he stays in etc. in a transparent and unobtrusive manner. The gathered data thereon gets employed to analyse the user's interruptability and thus can be used to decide on an appropriate action, e.g. should the call be forwarded to the user or rather be suppressed.

[4] The term „Artificial Receptionist“ has been borrowed from [Vorburger 05]

2.2 Key requirements

To achieve the result portrayed in our foregoing scenario, namely an application officiating as an artificial receptionist able to decide which phone calls to forward to the user based on information about its current context, we need to fulfil or at least approximate a couple of key requirements subsequently specified.

2.2.1 Unobtrusive application

The application constantly gathers data from its sensors. In order not to disturb the user by forcing him to interact with it, the application should run as a background process. Moreover, an application running in the background allows for an unbiased collection of data, increasing their significance for interruptability inference.

Additionally, in order to conserve storage capacity on the expansion card, sensor data should be gathered in a temporary buffer during a predefined time-frame before an incoming call occurs[5]. Within this time-frame the data streams should have the continuing ability to gather data, even if the user starts a different application. If an interruption occurs, thus stopping the gathering of sensor data, the application has to resume the recording by itself and without perceivable indications.

2.2.2 Intercept incoming calls

In order to manage and control incoming calls, our application must have the ability to intercept them before the operating system handles them by default. This step allows our application to superimpose additional queries such as a triggered sampling process, described in more detail in [Fornallaz 04] and in part implemented in [Donner 05], whereby the user gets confronted with multiple dialogs regarding the disturbance level of a call. Before answering the phone, the user has to estimate the importance of the call based on the caller's identity, followed by accepting or rejecting the call itself. After the

[5] The exact amount of time needed to gather sufficient data for later interruptability inference has yet to be defined. In [Fornallaz 04] the recording time for an audio file has to last at least 5 minutes, which could lead to an overall recording time-frame of about 15 minutes.

conversation a dialog appears prompting the user to rate the actual conversation regarding its effective importance.

2.2.3 Sensing the context

Trying to predict a user's context requires the collecting and evaluation of as many data sources as possible, since neither the prediction relevance of the individual sensors nor the algorithms needed for inference are currently known. We thereby have to tap all sensors capable of providing potentially useful information on the smartphone itself (so-called phone internal sensors) and on third party devices (so-called phone external sensors)[6].

In order to collect the data from various phone internal and external sensors, we need to implement an application capable of receiving and storing the information safely and reliably. The precondition thereof consists in exploring the types of data that can be acquired by a smartphone. The following subsections describe these sensors in more detail.

[6] From now on the terms "data source" and "sensor" will be used synonymously.

2.2.3.1 Internal sensors

As in part enumerated in [Fornallaz 04] following internal data sources are useful indicators for interruptability.

- **Cell ID**
  The Cell ID allows to detect the network cell in which the user currently is located. Since the authorization to retrieve and use this information depends on the respective telecommunications provider, an integration of this attribute has to be considered as the case arises.

- **Transmission signal strength**
  The transmission signal strength roughly measures the distance between the device and the nearest antenna tower. The information gained by this value, used concurrently with the Cell ID, permits to find out approximately where the device resides inside a cell and whether it will shortly switch to an adjacent cell. Applied to our framework, the combination of the signal strength, coupled with the cell ID and some complementary information basically suffices to identify a frequently attended location, such as one's office workplace, with a high degree of accuracy.

- **Calendar entries**
  The integration of calendar entries, most notably as supplementary information during a phone call, allow to incorporate possible appointments into the evaluation of the user's interruptability.

- **History log**
  The history log consists of a queue or array in which all user actions on a smartphone are recorded, thus allowing to gather statistical data on various device features for additional interruptability inference. Although artificially created, it has all the properties characteristic of an internal sensor.

- **Date and time**
  The attachment of date and time values to gathered data allows for a chronological sorting of the recorded sessions.

- **Sound recording**

  Audio data gets collected in the form of a stream, which continually stores information in a wave file on an expansion card of the smartphone. Depending on the desired sound quality and bandwidth constrictions, properties such as modulation and sampling rate can be adapted to suit one's needs The characteristics of the audio stream will be covered in more detail in the implementation section.

- **Still images**

  This internal sensor has its use in spotting motion detection by means of frame differencing algorithms as described in [Zurfluh, 04]. With still images, the only possibility to detect motion is given by taking at least two consecutive snapshots. Another purpose of image capturing is the measurement of brightness. The brightness level of the image can offer clues as to where the device is located. The respective brightness values vary depending on whether the individual has put the device in a pocket or left it on a table [Fornallaz 04]. These findings, as used in the SenSay project [Siewiorek 03], provide a basis to decide whether the user should be interrupted [de Simoni 05].

2.2.3.2 External Sensors

The external sensors consist of a GPS receiver and a custom made sensorboard with a wide array of sensors on it. These external data sources, or external sensors, are connected to the smartphone on the one hand via Bluetooth (in case of the GPS receiver) and on the other hand by a serial connection through the multi-connector interface. Figure 1 depicts the smartphone with attached external sensorboard.

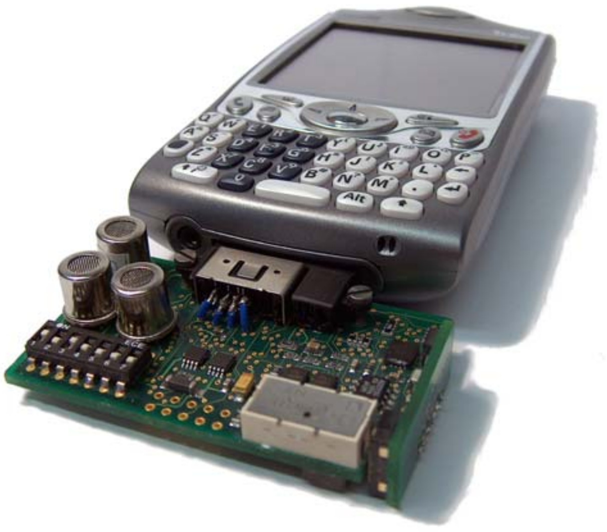


Figure 1: The Treo 650 smartphone with attached sensorboard

- **GPS receiver**[7]
  The global positioning system (GPS[8]) receiver allows to detect a user's exact location and hence his surroundings. The external GPS device transmits its data via the wireless Bluetooth protocol to the smartphone. Devices using Bluetooth do not have to be pointed at one another, the distance between devices can reach 100 metres, the speed of transmission up to 2.1 Mbit/s, and the data passed can be encrypted and secure. Despite these powerful features, the battery consumption is relatively low[9]. For further information regarding Bluetooth consult https://www.bluetooth.org/ and [Roth 02].

[7] Detailed information about the Socket GPS receiver can be found in Appendix B.
[8] The Global Positioning System is a satellite navigation system used for determining one's precise location and providing a highly accurate time reference almost anywhere on Earth or in Earth orbit. It uses an intermediate circular orbit (ICO) satellite constellation of at least 24 satellites. http://en.wikipedia.org/wiki/Gps (Accessed December 2005)
[9] http://www.palmos.com/dev/tech/bluetooth/ (Accessed December 2005)

- **Sensorboard**

  The sensorboard is an experimental prototype board containing a full set of sensor chips. The sensor array is composed of a 3-D accelerometer measurement unit **[1]**, a 3-D gyroscope measurement unit **[2]**, a magneto-resistive sensor **[3]** able to create a digital magnetic compass, three gas measurement units **[4]** capable of detecting trace amounts of carbon monoxide (CO) / methane (CO_4), alcohol and liquefied petroleum gas respectively, and a temperature measurement unit **[5]**.

  In addition, the board features an RS-232 compliant interface capable of communicating with various devices such as Notebooks, PDAs and of course, smartphones. Coupled with the GPS client, the sensorboard might allow the extraction of navigation relevant information from people's motion data as specified in the approach adopted by [Tschanz 05].

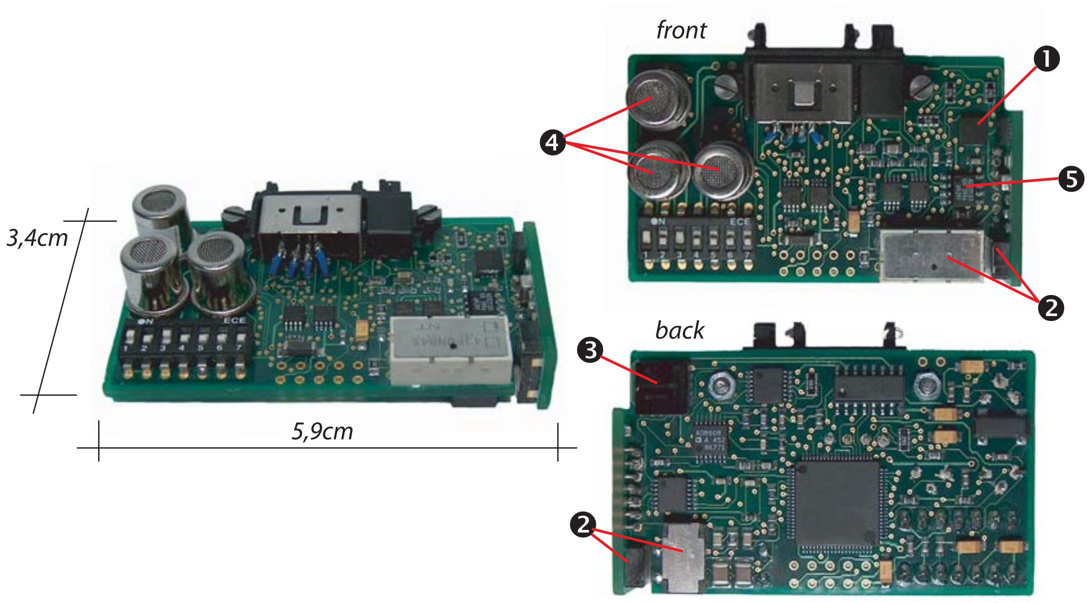


Figure 2: The sensorboard with muti-connector for the Treo 650

2.2.4 Parallel processing of the data streams

The data generated by both internal and external sensors has to be received and processed in parallel. In other words, our application must have the ability to manage all incoming streams simultaneously. Since most internal sensors, such as Cell ID, signal transmission strength etc., provide individual values, it is a straightforward matter to retrieve them shortly before an incoming call and store them on the expansion card. The data streams, namely the audio, the GPS and the sensorboard stream on the other hand need to continuously gather data, all at the same time. Since both the GPS and the sensorboard stream are dependent on a serial connection to receive data, it is crucial that the serial manager is able to handle multiple connections.

2.2.5 Data logging / reasoning

To be able to gather and analyse the data later on, it is crucial to correctly integrate the sensors and to permanently store its values. As we will see in the implementation phase, all the relevant data can be safely stored on an expansion card, located on the device itself. The size of the storage space on such an expansion card can reach several Gigabytes, thus allowing the gathering of large amounts of data[10]. The data thereafter gets transferred to a desktop machine, where computationally expensive machine learning algorithms sift through it, trying to detect patterns useful for interruptability inference[11].

[10] Due to resources constraints on contemporary smartphones, our application is concerned purely with the logging of data. Online reasoning is therefore currently inapplicable.
[11] For further information concerning reasoning and data mining consult [Donner 05].

Chapter 3

Implementation

This chapter takes up the requirements mentioned in the previous chapter and analyses them from a technical point of view. The emphasis thereby lies on the implementation of various applications capable of gathering and storing data streams originating from internal and external sensors located on a smartphone such as, in our case, the Treo650 from Palm Inc[12].

After a brief overview of the general architecture of a Palm OS application follows an in-depth description of the structure and workings of the applications used to control both internal and external sensors. To tie in with the comprehensive description in the requirements chapter, section 3.2 will give an overview of the related work already done by other students to show the reader which part of the overall architecture these applications belong in.

[12] http://www.palm.com/us/products/smartphones/treo650/ (Accessed December 2005)

3.1 Generic architecture of a Palm OS application

Writing an application for smartphones running on Palm OS requires a different approach in comparison to desktop applications. On the one hand the user interacts differently with the smartphone, having a limited amount of time to expect an output from the device. It is therefore crucial to develop applications with preferably short execution times and intuitive user interfaces. In addition there is the issue of power and memory to be considered, both of which are available only in short amounts on the device itself.

Having said that, let us proceed to the actual setup of a Palm OS application. A Palm OS application does not have a start and exit command like a desktop application. It rather gets accessed by so-called launch codes. These launch codes can either be triggered by an explicit user request or in response to some other user action. An additional way to launch an application is given by so-called notifications, which react on certain system-level events or application-level events. In order to make use of notifications the application must be specifically registered to receive them. Instead of the nonexistent exit command, a Palm OS application terminates when the user requests another application.

3.1.1 The event manager

The main interface between the Palm OS system software and the application consists of the event manager. The event manager is responsible for the behaviour the application displays in answer to user input or system events, thus making it the key administrator of the running application. The modus operandi of the event manager can be described as follows: after receiving the launch code the application starts with a startup routine, followed by the event loop and finally exiting with the stop routine. The event loop is thereby responsible for administering the fetching and dispatching of events, taking advantage of the default system functionality as appropriate. Events comprise all system level events and user interactions which force the operating system to respond to them. The pen tap on a form is an example of a user event. Figure 3 exemplifies a control flow in a typical application. [13]

[13] http://www.palmos.com/dev/support/docs/palmos/PalmOSCompanion/EventLoop.html#1004923 (Accessed December 2005)

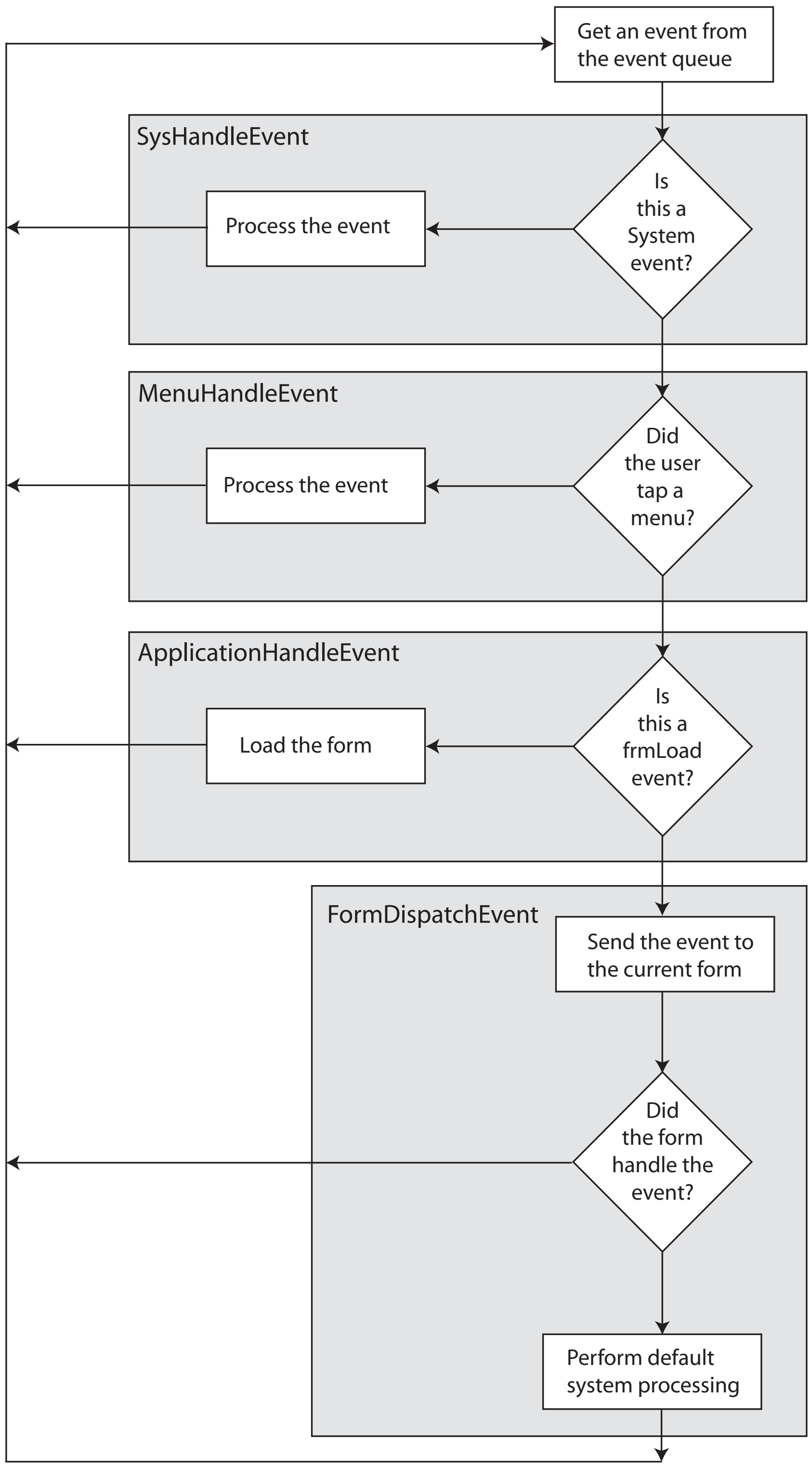


Figure 3: A control flow in a typical application

3.2 Related work

An exhaustive survey on related work regarding the field of context awareness and mobile technology in general is given in [Fornallaz 04] and [de Simoni 05]. This section focuses on antecedent work directly related to our context sensing application.

In order to conduct the experiment leading to our working scenario portrayed in chapter 2, we have to gather all kinds of data originating from various sensors located both on the smartphone itself and on external devices connected to the smartphone via various protocols. After having collected and safely stored the data on the handheld, it gets relocated to a desktop machine, where the preprocessing takes place.

The first concept of an application acting as a centralized recorder of various data streams during an ongoing sampling session can be found in [Fornallaz 04]. It is built upon the standard Symbian application scheme and mainly allows to store the data in a folder containing a timestamp and metadata required for the sampling session. Figure 4 provides an insight into the context sampling procedure. The device hosting the application consisted of the Sony Ericsson P800 based on Symbian OS7.

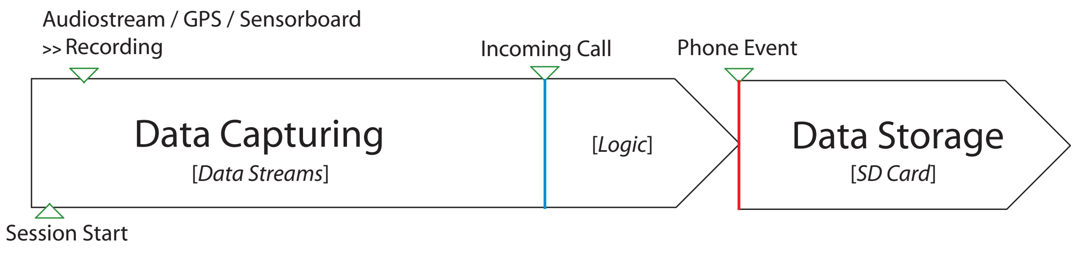


Figure 4: Sampling session sequence

Due to constraints regarding hardware limitations and fragmentary API support, both the underlying device and thus the operating system had to be changed. After testing several smartphone manufacturers and the operating systems their devices were based on, the choice fell on the Treo 600 from Palm Inc. and thus on the Palm OS. In [deSimoni 05] the context recorder architecture has been ported to the Palm operating system, complementing the functionality missing in the previous version. Despite the improved support of the new operating system a few key features, such as the audio stream and the Bluetooth protocol, could still not be implemented, partially because of missing API and/or hardware support on the test device itself.

The main focus of the following section consists in presenting the applications responsible for the integration of newly added sensors such as the internal sensors in general, the GPS client, the sensorboard and the audio recorder.

3.3 External Sensors

The array of external sensors comprises the proprietary sensorboard and the GPS receiver. This section addresses the design and implementation of applications capable of gathering and storing data deriving from these sensors. The Palm OS serial manager, on which both data streams rely on, will also briefly be introduced.

3.3.1 The serial manager

An important aspect of the Palm OS architecture regards the serial manager, since both the sensorboard and the Bluetooth stream depend on it. The Palm OS serial manager is responsible for byte-level serial I/O and control of the RS-232, IR, Bluetooth or USB signals. It additionally manages multiple serial devices with minimal duplication of hardware drivers and data structures. This feature makes it possible to bundle our data streams in one application. Figure 5 shows the layering of communication software with the serial manager and hardware drivers.[14]

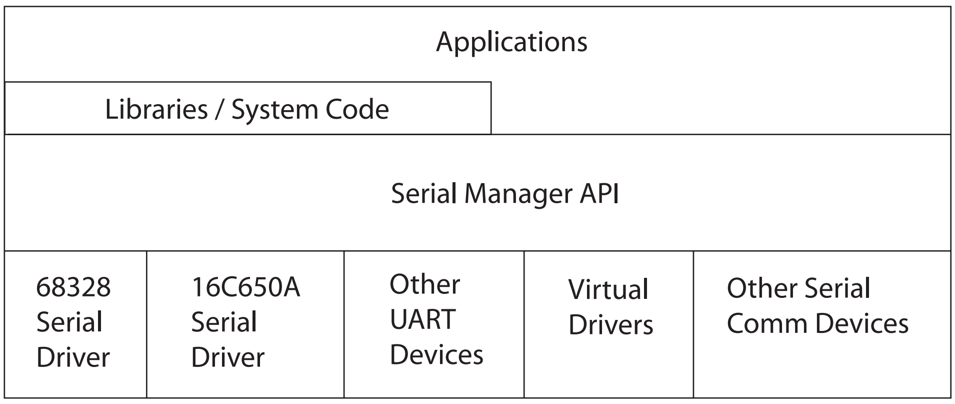


Figure 5: Layering of communication software

[14] http://www.palmos.com/dev/support/docs/palmos/PalmOSCompanion2/SerialCommunication.html (Accessed December 2005)

3.3.2 Serial Port

The most crucial data stream to be integrated consists of values deriving from the sensors on our custom made board. The board is connected to the smartphone through a multi-connector using serial communication via a RS-232 connection[15] as depicted in figure 6.

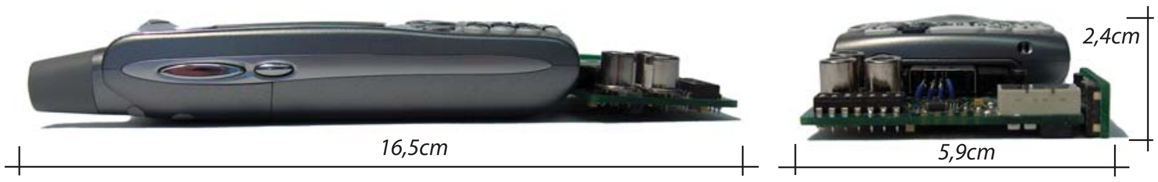


Figure 6: The sensorboard attached to the smartphone via the multi-connector

To correctly receive a data stream over the serial port Palm OS requires a number of conditions to be fulfilled, namely the specification of an interface, the definition of a baud rate[16] and the explicit opening and closing of the serial port itself. The syntax to open the port looks as follows:

```
// open port
#define SERIAL_PORT serPortCradleRS232Port
err = SrmOpen( SERIAL_PORT, gBaudRate, &gPortId );
```

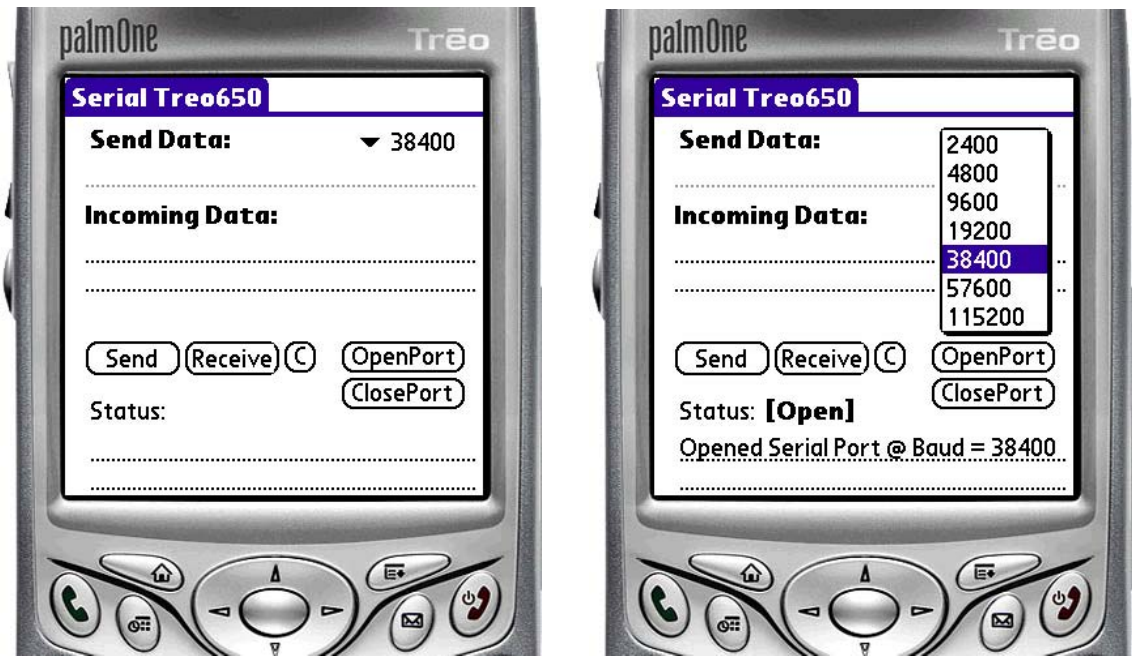
The baud rate can be set to one of the following values depending on bandwidth consumption: 2400, 4800, 9600, 19200, 38400, 57600, 115200. After careful adjustments the baud rate is currently set at 38400. This setting allows a clean data transmission even when the bandwidth consumption reaches full capacity, e.g. when all data streams are activated. After a brief negotiation routine on both the smartphone and the sensorboard, a connection gets established. The ensuing data stream flows into the smartphone where it gets buffered and gradually forwarded to a text file. The buffering procedure uses a time-constrained window to transfer data into a variable located on the smartphone itself. Experiments however have shown, that the only relevant factor which influences the

[15] RS-232 is a standard for serial binary data interconnection between a DTE (Data terminal equipment) and a DCE (Data communication equipment). It is commonly used in computer serial ports. http://en.wikipedia.org/wiki/RS-232 (Accessed December 2005)
[16] baud is a measure of the "signaling rate" which is the number of changes to the transmission media per second in a modulated signal. It is named after Émile Baudot, the inventor of the Baudot code for telegraphy
http://en.wikipedia.org/wiki/Baud_rate (Accessed December 2005)

speed of the data transfer, is comprised of the baud rate itself. The simplified buffering procedure looks as follows:

```
err = SrmReceiveWindowOpen( gPortId, &serialBufferP,
&numBytesWindow );
  freeSpace -= numBytesWindow;
  if (numBytesWindow > 0) {
    MemMove( receiveBufferP, serialBufferP,
(Int32)numBytesWindow );
  }
err = SrmReceiveWindowClose( gPortId, numBytesWindow );
```

The buffered data gets displayed on the screen of the smartphone to allow an examination of the incoming sensor values. A text field, which relays user input to the sensorboard furthermore permits to switch specific sensors on and off on the board itself. The changes in values are displayed in real time.



Opened Serial Port @ Baud = 38400

Figure 7: The serial application establishing a connection with the sensorboard

While being displayed on screen, the data additionally gets rerouted onto a text file on an expansion card for permanent storage and further analysis on desktop platforms. The raw data consists of an array of twelve triple-digit hexadecimal values representing the various sensor readings such as temperature, acceleration etc. as depicted in table 1. For an exhaustive description of the capabilities of the sensorboard consult [Tschanz, 2005].

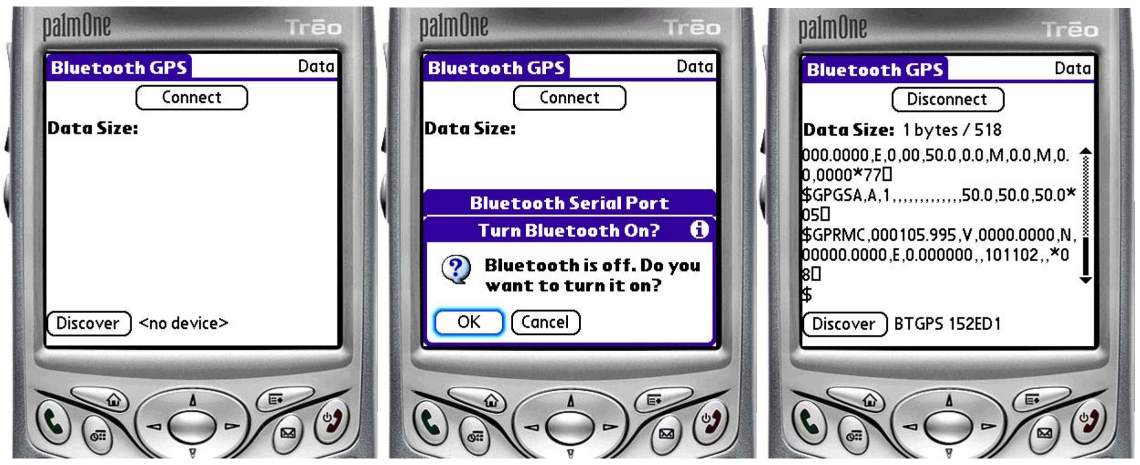
```
FFA,F9C,FAA,FAB,88F,A5B,A4C,1FF,1FC,FFB,00E,009
FFB,A58,A47,1FF,207,005,003,009,FA1,FB5,879,A5A
FFA,FB0,F9A,FA6,85A,A4F,A49,1F6,1EC,00B,004,00D
FFB,003,00A,A48,1EF,1FB,011,FFB,FFB,FB0,850,A4D
FFA,FA1,FC0,FBF,859,A4A,A3D,1EE,1FA,003,011,FFB
FFB,00A,002,84F,A4E,A3C,1F4,1FF,008,009,00B,A50
FFA,F9D,FA5,FA2,82D,A3F,A3E,1F8,1FF,005,011,A46
FFA,FB0,F9A,FA6,85A,A4F,A49,1F6,1EC,00B,004,00D
FFB,003,00A,A48,1EF,1FB,011,FFB,FFB,FB0,850,A4D
FFB,A58,A47,1FF,207,005,003,009,FA1,FB5,879,A5A
FFB,003,00A,A48,1EF,1FB,011,FFB,FFB,FB0,850,A4D
FFA,FA1,FC0,FBF,859,A4A,A3D,1EE,1FA,003,011,FFB
FFB,00A,002,84F,A4E,A3C,1F4,1FF,008,009,00B,A50
FFA,F9D,FA5,FA2,82D,A3F,A3E,1F8,1FF,005,011,A46
FFB,A58,A47,1FF,207,005,003,009,FA1,FB5,879,A5A
FFA,FB0,F9A,FA6,85A,A4F,A49,1F6,1EC,00B,004,00D
FFB,003,00A,A48,1EF,1FB,011,FFB,FFB,FB0,850,A4D
```

Table 1: Hexadecimal sensorboard output

The serial application additionally features a status field which notifies the user of the connection status and, if connected, of the amount of data transferred. On top of that it informs the user with customized error messages of possible connection problems.

3.3.3 Bluetooth GPS

The integration of the GPS signal via Bluetooth has a similar implementation as the one on the sensorboard. Both actually rely on the serial manager architecture. To successfully integrate the GPS stream though the additional Palm OS Bluetooth SDK had to be used. The SDK allows access to the certified SIG Bluetooth stack, thus allowing for a device independent implementation of applications using this technology. Another fundamental difference between the RS-232 and the Bluetooth connection is the master-slave architecture Bluetooth uses to negotiate a communication between two devices. Furthermore a device discovery is launched for first-time connections. This allows the application to differentiate between trusted and merely connected devices, therefore consequently increasing security[17].



Data Size:

<no device>

Bluetooth Serial Port

Turn Bluetooth On?

Bluetooth is off. Do you want to turn it on?

Data Size: 1 bytes / 518

```
000.0000,E,0,00,50.0,0.0,M,0.0,M,0.
0,0000*77
$GPGSA,A,1,,,,,,,,,,,,,50.0,50.0,50.0*
05
$GPRMC,000105.995,V,0000.0000,N,
00000.0000,E,0.000000,,101102,,*0
8
$
```

BTGPS 152ED1

Figure 8: GPS connection procedure and incoming data stream

The application acts as the Bluetooth master device and is capable of launching a discovery to detect nearby slave devices. Furthermore it is able to differentiate between trusted and untrusted devices[18]. Only after the GPS device has successfully been identified as slave, can a connection be established. The procedure bears resemblance to the sensorboard connection as following source code demonstrates.

[17] In an automated data gathering application, the device discovery can be omitted by inserting the Bluetooth device’s signature into the sourcecode directly.

[18] Devices and services have different security levels. For devices, there are 2 levels, "trusted device" and "untrusted device". A trusted device, having been paired with one’s other device, has unrestricted access to all services.

```
// Find and open the Bluetooth library
if( SysLibFind(btLibName, &gBtLibRefNum) ) {
  err = SysLibLoad(sysFileTLibrary, sysFileCBtLib, &gBtLibRefNum) ;
    if(err) {
      ErrFatalDisplay("Unable to load the Bluetooth stack");
      return err;
    }
}
err = BtLibOpen(gBtLibRefNum, false);
gBtLibOpen = true;

// Read the data from the serial port
receivedBytes = SrmReceive (gBtPortId, (void *)bufP, count, 10,
&err);
if (!receivedBytes) {
  MemPtrFree(bufP);
  return;
}
```

The ensuing data stream gets both displayed on screen and stored onto a text file on the expansion card analogue to its sensorboard counterpart. The data stream makes use of the NMEA[19] 0183 standard, which is a standard protocol used by GPS receivers to transmit data. The output is RS-232 compatible and transmits data with 4800 bps, 8 data bits, no parity and one stop bit. Sentences beginning with the `$GP` token are used to identify GPS data as illustrated in table 2.

```
$GPGGA,000020.998,0000.0000,N,00000.0000,E,0,00,50.0,0.0,M,0.0,M,0.0,0000*7C
$GPGSA,A,1,,,,,,,,,,,,,50.0,50.0,50.0*05
$GPRMC,000020.998,V,0000.0000,N,00000.0000,E,0.000000,,101102,,*03
$GPGGA,000021.998,0000.0000,N,00000.0000,E,0,00,50.0,0.0,M,0.0,M,0.0,0000*7D
$GPGSA,A,1,,,,,,,,,,,,,50.0,50.0,50.0*05
$GPRMC,000021.998,V,0000.0000,N,00000.0000,E,0.000000,,101102,,*02
$GPGGA,000022.998,0000.0000,N,00000.0000,E,0,00,50.0,0.0,M,0.0,M,0.0,0000*7E
$GPGSA,A,1,,,,,,,,,,,,,50.0,50.0,50.0*05
$GPRMC,000022.998,V,0000.0000,N,00000.0000,E,0.000000,,101102,,*01
$GPGGA,000023.998,0000.0000,N,00000.0000,E,0,00,50.0,0.0,M,0.0,M,0.0,0000*7F
$GPGSA,A,1,,,,,,,,,,,,,50.0,50.0,50.0*05
$GPRMC,000023.998,V,0000.0000,N,00000.0000,E,0.000000,,101102,,*00
$GPGGA,000024.998,0000.0000,N,00000.0000,E,0,00,50.0,0.0,M,0.0,M,0.0,0000*78
$GPGSA,A,1,,,,,,,,,,,,,50.0,50.0,50.0*05
$GPRMC,000024.998,V,0000.0000,N,00000.0000,E,0.000000,,101102,,*07
```

Table 2: GPS receiver output

[19] NMEA stands for **N**ational **M**arine **E**lectronics **A**ssociation. The official website can be found at http://www.nmea.org/. A comprehensive syntax description of the NMEA 0183 standard is located in http://www.kh-gps.de/nmea.faq (Accessed January 2006).

3.4 Internal Sensors

Internal sensors are features located on the smartphone itself, adapted to provide information in addition to the data sources deriving from the external sensors. This section describes the applications developed to retrieve and store the relevant information.

3.4.1 Sound stream

The Treo smartphone has a two-speaker audio architecture. One speaker, called the receiver, is mostly dedicated to telephony sound and is tuned to voice frequency. The other, the external speaker, is mostly dedicated to system sounds and is tuned to polyphonic sounds [palmOne 04]. In our case we have modified the receiver to act as a recording device. The structure of the recording application resembles a conventional recorder. It has the ability to start a recording, to pause it and to stop it completely. The ensuing sound stream gets stored in a file with the `.wav` ending on the expansion card alongside the other data streams. The parameters defining the quality of the recorded streams can be chosen from a wide array of configuration options such as the choice of modulation, a mono-stereo selector, bitrate etc. These settings will be discussed in more detail later on in this chapter.

The design of the application is quite straightforward considering the rather intricate implementation underneath. It consists of a button to start a recording of the audio stream. A timer keeps track of the already elapsed time, while the stop button halts the recording and closes the audio stream. Figure 9 shows the audio stream controls.

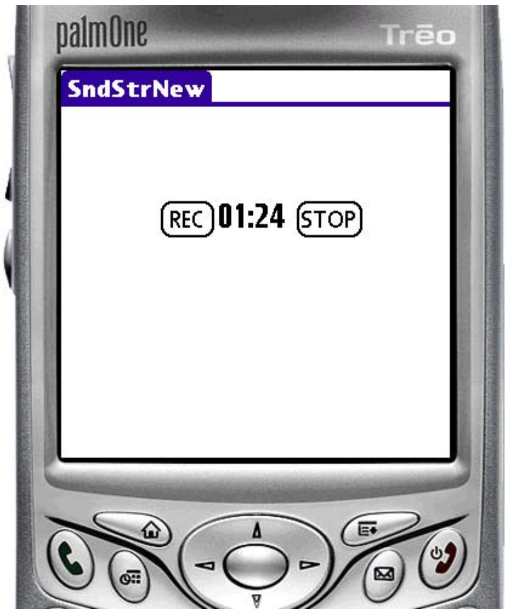


Figure 9: Soundstream recorder during a recording session

The Palm OS sound manager API offers mono and stereo recording capabilities. Important to keep in mind though is the fact that a stereo recording automatically doubles the data amount to be stored on the expansion card while only marginally improving the recording itself.

The sample type and sample rate are two additional parameters allowing to adjust the recording quality. The sample type can be set to 8bit or 16bit. The higher the bit depth the more subtler changes in volume become audible. The sample rate on the other hand defines the amount of samples that are being recorded per second[20]. The unit is measured in kHz (thousand samples per second). A formula able to calculate the transfer rate of an audio stream in Kilobytes per second is given in figure 10.

$$KByte/s = \frac{SampleRate\left[s^{-1}\right] \cdot SampleType\left[bit\right]}{4 \cdot 1024\left[KByte \cdot bit^{-1}\right]}$$

Figure 10: Audio stream formula

[20] A sample is the smallest amount of information in a digital recording. It corresponds to a value of the analog signal.

These settings can easily be altered by changing the few key variables located in the global variables section of the main application file. In case of the *SerBTSound* application the main file is `SerBT.c`. A listing of all modifiable sound stream parameters is outlined in the `PlayerInfo` struct[21].

```
typedef struct _PlayerInfo {
  UInt32          sampleRate;     // 8000/11025/16000/...
  UInt16          blockAlign;     // Mostly there for ADPCM
  UInt32          dataSize;       // Size of the data chunk(in bytes)
  UInt32          bufferSize;     // Requested buffer size
  UInt32          actualSize;     // Real buffer size
  Int32           streamVolume;   // Volume [0..2048]
  SndStreamMode   streamMode;     // Input/Output
  SndFormatType   streamFormat;   // sndFormatPCM/sndFormatIMA_ADPCM
  SndStreamWidth  streamWidth;    // sndMono/sndStereo
  SndSampleType   sampleType;     // UInt8....
  SndStreamRef    streamRef;      // Our stream reference

  Char            fileName[64];   // File name
  FileType        fileType;       // VFS/Memory
  union {
    FileRef       fileRef;        // File ref for VFS
    FileHand      fileHandle;     // File Handle for File Streaming
  };

  UInt16          volumeRef;      // Card ref
  Char            directory[256]; // The SD/MMC card audio directory
  PlayerState     playerState;    // IDLE/PLAYING/PAUSED...
} PlayerInfo;
```

Table 3: Sound stream struct definition

[21] The struct keyword is used to declare a new data-type by means of grouping variables together.

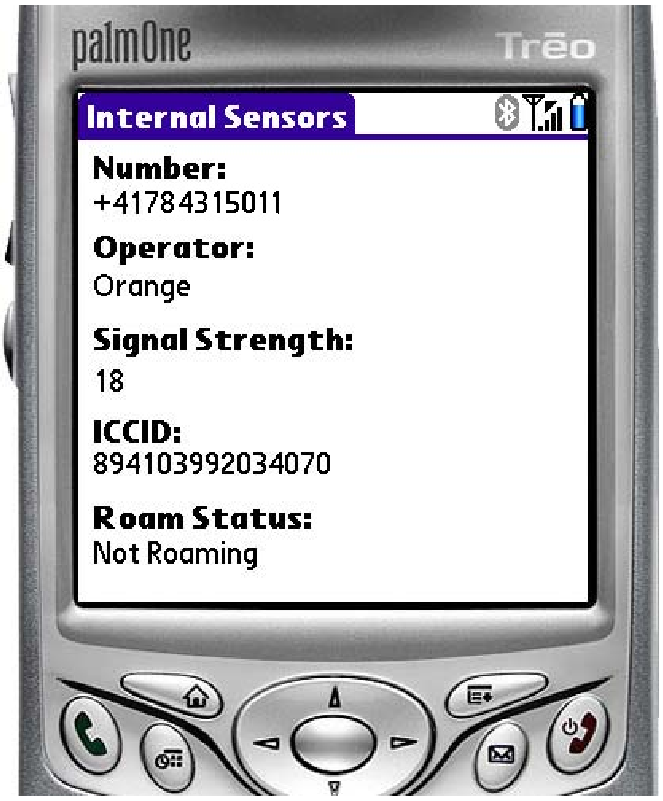
3.4.2 Remaining Internal Sensors

Palm Inc.’s `PhnLib` API library allows to retrieve a wide array of information concerning the status of the smartphone’s connection to the network. Since these sensors do not have to gather data continually like streams, they can be accessed and stored just before or after an incoming call has occurred. Figure 11 depicts an application capable of retrieving parameters from the network in real-time[22]. For simplicity’s sake it is called *InternalSensors* and is composed of a main screen displaying the requested network information according to the parameter list provided.

At first the application checks which cellular technology the network uses, GSM or CDMA. CDMA, short for **C**ode-**D**ivision **M**ultiple **A**ccess, is a digital cellular technology that uses spread-spectrum techniques. It is a method of multiple access that does not split up the channel by time (as in TDMA), or frequency (as in FDMA), but instead encodes data with a certain code associated with a channel. GSM on the other hand uses TDMA. Short for **T**ime **D**ivision **M**ultiple **A**ccess, TDMA is a technology for delivering digital wireless services using time-division multiplexing (TDM). It works by dividing a radio frequency into time slots and then allocating slots to multiple calls. In this way, a single frequency can support multiple, simultaneous data channels. CDMA competes with GSM technology for dominance in the cellular world. After the network identification, the application retrieves the wanted data from the mobile provider and displays it on the main screen[23].

The actual parameter selection, displayed on the application’s main screen, can be expanded arbitrarily by additional values provided by the API and according to possible requirement changes[24].

[22] As of this writing, the Palm Inc. API does not provide a function able to retrieve the Cell ID as mentioned in the requirements section.
[23] In the integrated application, there will be no need to display the values on screen. The data will be collected and directly stored on the expansion card just before or after a call has occurred.
[24] For an exhaustive list of available network parameters consult the `PhnLib` library in Palm Inc.’s API reference guide.



Number:
+41784315011
Operator:
Orange
Signal Strength:
18
ICCID:
894103992034070
Roam Status:
Not Roaming

Figure 11: The Internal Sensors application

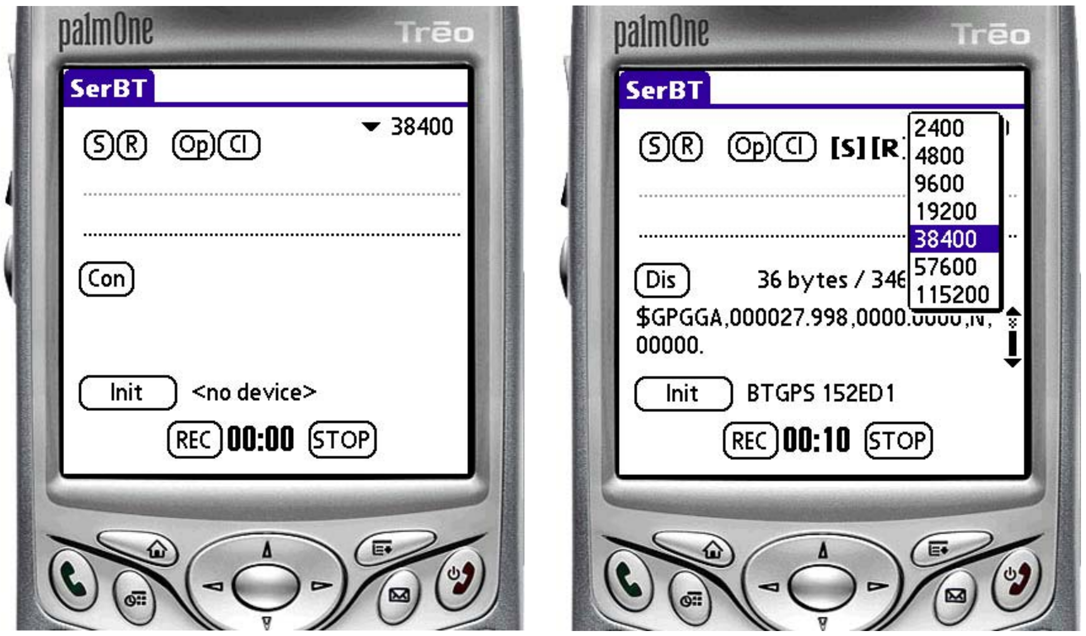
3.5 Parallel processing of the data streams

After successfully having integrated the external as well as the internal sensors with their respective data streams in standalone applications, the crucial part now consisted in merging the streams into a single application, thereby allowing it to gather data in a concurrent fashion. To do that, each standalone application had to be transformed into a module capable of interacting with the remaining ones. The next step consisted in providing a uniform interface connecting the module cluster with the underlying serial and event managers of the operating system.

Both the serial and the event manager are able to process the data streams concurrently and to write the data on the expansion card in real-time, thus making the integration project a success.

The resulting application is called *SerBTSound*[25] and consists of a graphical user interface (GUI), which allows to control the recording activities of all data streams. It additionally displays the incoming data in text fields for the user to see, thus allowing a closer monitoring of the received values, especially the ones originating from the sensorboard. From a graphical point of view the integration of the individual data streams into a single GUI was a straightforward matter. The respective standalone GUIs had to be stripped of any unnecessary elements to be able to fit on one single screen measuring 160x160 pixel as figure 12 illustrates.

[25] The name is a composite of the main data stream labels ***Se****rial*, ***B****lue****t****ooth* and ***Sound***.



▾ 38400

<no device>

00:00

36 bytes / 346

$GPGGA,000027.998,0000.0000,N,
00000.

BTGPS 152ED1

00:10

Figure 12: GUI of all integrated data streams

3.6 Constraints and possible workarounds

Even though the Palm operating system has evolved and the new device offers some new features, there are still unresolved issues that need to be addressed in future work. The most relevant new feature is the ability of the Treo 650 to exchange data via the widespread Bluetooth protocol, thus allowing for the integration of our GPS client. The improved APIs of Palm OS version 5.4 furthermore allowed to successfully embed the internal audio stream and the external sensorboard stream via the serial connection.

3.6.1 Unobtrusive application

A few key requirements still could not be implemented as planned in the conceptualization phase. One of the specifications regards the execution of the application as a background task. This requirement implies a multithreading architecture, which the current Palm OS is not able to provide[26]. Even though the underlying hardware supports multithreading, there are no APIs available to the programmer. Only the upcoming operating system from Palm Inc, codenamed Cobalt is capable of multithreading and multitasking. The transition to this new operating system however will be a hesitant one[27]. Currently the only company to produce a smartphone based on the Palm OS Cobalt platform looks to be a Singapore company with its Zircon Axia A108[28].

To bypass this constraint, a possible workaround would consist in setting up a cleverly devised structure of system and event notifications allowing to approximate a background process with as few interruptions in the data gathering process as possible. Since we can not execute two applications simultaneously, we need to have a mechanism in place that

[26] As of this writing, the current Palm OS version running on the Treo 650 is Garnet v5.4. http://www.palm.com/us/products/smartphones/treo650/details.epl (Accessed December 2005)
[27] Palm’s President Ed Colligan refused to commit to producing Palm Cobalt devices in 2005. In an interview with NewsWireless.net, he states "Nobody knows when we'll start the shift to Cobalt, OS 6, or on which devices. For now, we're saying that we've built the functionality we need into the Treo and the Tungsten T5 and there's no need to confuse developers by switching. I'm not even prepared to commit us to a change next year, or the year after, at this stage." http://www.palmzone.net/modules.php?name=News&file=article&sid=298 (Accessed December 2005).
[28] http://www.smallbizpipeline.com/164301458 (Accessed December 2005)

consistently restarts our data gathering tool each time an interruption occurs. Event notifications coupled with specific functions able to switch between applications autonomously allow the build-up of an approximated background process. A more exhaustive outlook into this framework is given in appendix C.

3.6.2 Photo recording

An application to capture and store pictures has already been implemented. It is capable of capturing an image with a resolution of up to 640x480 pixels (VGA). The captured image then gets displayed on the smartphone’s screen. The only problem lies in the missing functionality to persistently store a captured image either on the expansion card or in the memory of the smartphone itself. Even after repeated attempts by means of different approaches, a satisfactory solution still has to emerge.

Using alternative image manipulation libraries, it could be possible to extract the raw data directly from the smartphone’s buffer, without the need to convert it to a standardized format, such as *jpg*, first.

Chapter 4

Measurements

- Audio
- GPS
- Sensorboard Charts
- Battery Consumption

This chapter addresses the properties of the collected and stored data pertaining to consistency and quality. In order for the data to be suitable for data mining later on, it has to be properly gathered and stored. The data then gets transferred to a desktop computer, where the preprocessing takes place. Preprocessing allows to conduct a feature extraction operation on raw audio, image and sensor data as described in [de Simoni 05][29]. The last section briefly touches the topic of battery consumption in order to estimate the durability of a continuous recording session.

4.1 Audio

In both their theses [Fornallaz 04] and [de Simoni 05] have exhaustively evaluated and analysed audio data by means of frequency and spectrometry analysis, thereby acting out various scenarios to determine the data's eligibility as a possible context analysis candidate. Audio data seem to be an adequate candidate for context analysis.

4.2 GPS data stream

The GPS client establishes a connection to the smartphone via the Bluetooth protocol stack. Since both the Bluetooth connection negotiation sequence and the GPS data originating from the device employ standardized send and receive procedures with integrated failsafe mechanisms preventing data loss, the quality and usability of the gathered data can be assumed to be very high and thus definitely usable.

[29] The data processing functions currently taking place on desktop machines should in the future be relocated onto the smartphone itself.

4.3 Sensorboard data charts

The sensorboard sends a continuous data stream through the serial connection without flow control mechanisms to regulate its output. It is therefore crucial that the receiving end of the stream, in our case the incoming serial buffer of the smartphone, is capable of correctly receiving and storing the incoming data. For an in-depth look regarding baud rates and balancing issues consult Appendix D.

In order to envision what the sensorboard values look like when arranged in an organized fashion, this section gives a visual foretaste of several graphs deriving from sensor data. The sensors have been specifically targeted to allow a corresponding reaction to manifesting itself on the graph.

The gas sensor for example, which measures the carbon monoxide and/or methane gas content in the air, has been subjected to a car ride around the city. The result can be observed in figure 13.

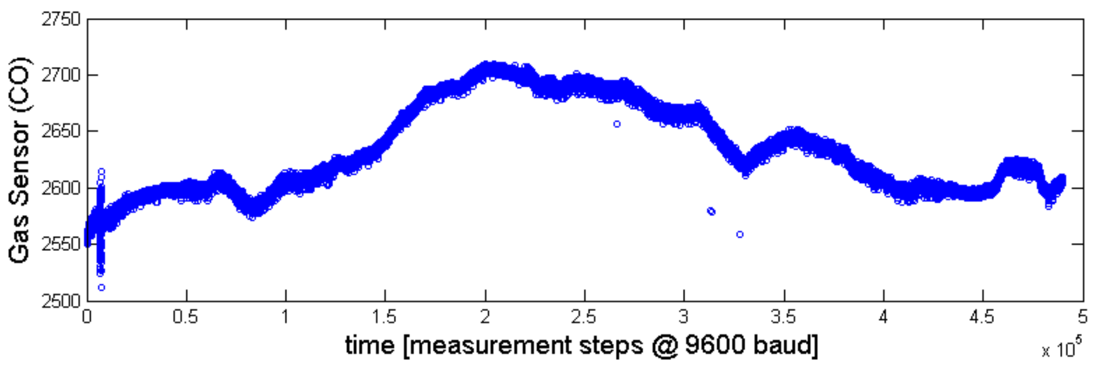


Figure 13: Gas Sensor

The temperature sensor on the other hand, has been tested with a soldering gun as primary heat source. The peaks in the graph clearly delineate the heating up of the sensor due to a threefold pass with the soldering gun.

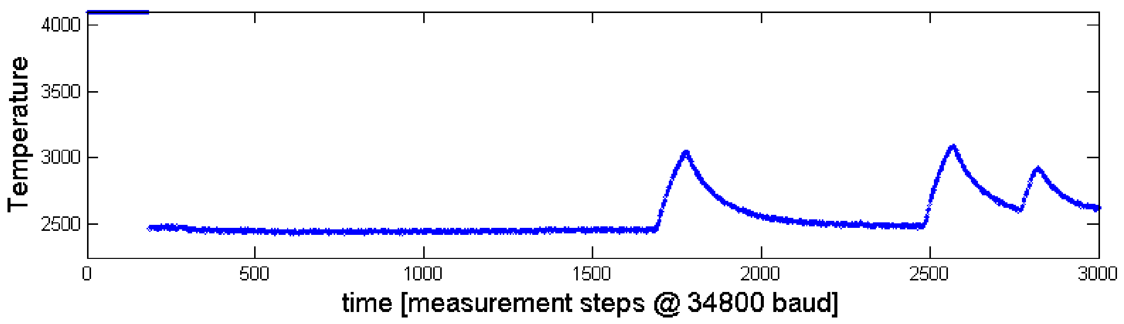


Figure 14: Temperature sensor

In order to try out the magnetic sensors, at first they have been subjected to a multitude of movements in different directions, recognizable on the first half of the diagram. Afterwards, a blackboard-magnet has been moved in sweeping motions over the sensor array at various speeds, thus explaining the jittery course of the second half of the chart.

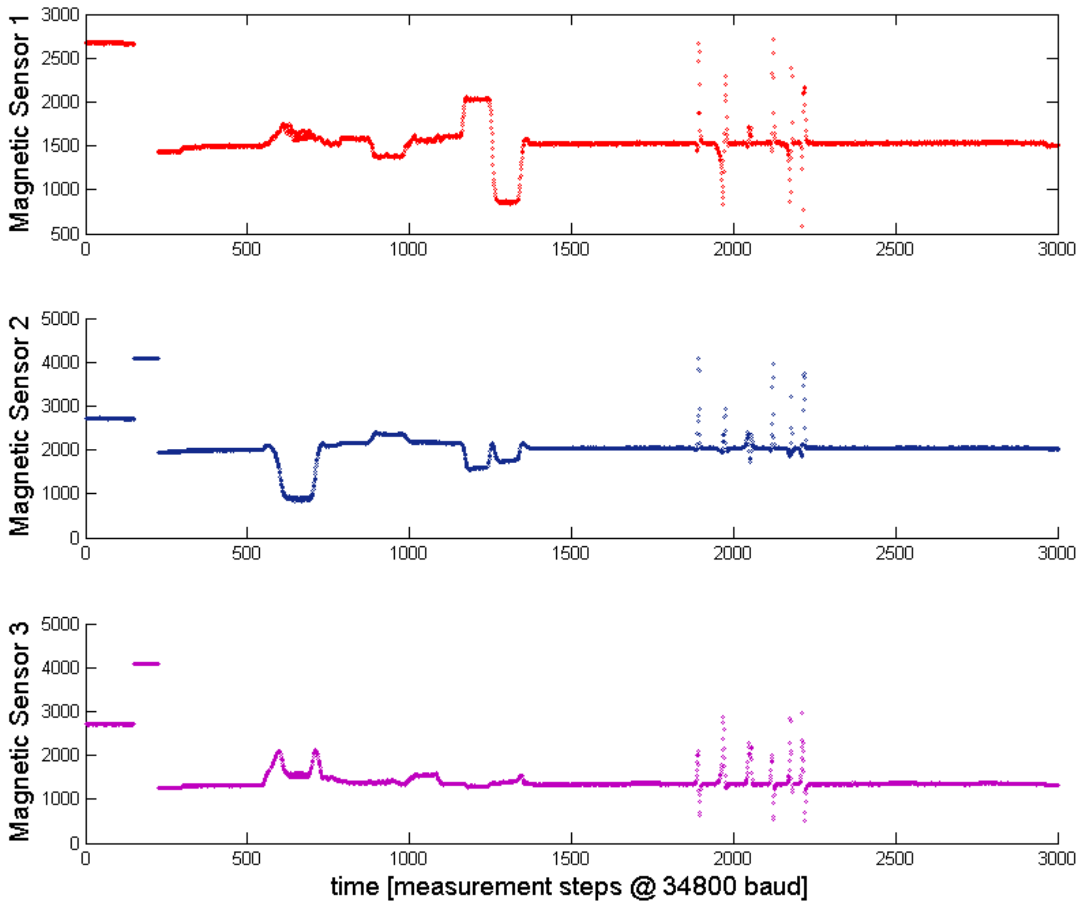


Figure 15: Magnetic Sensors

The gyroscopes[30] and the accelerometers[31] have undergone multiple motion tests; namely a circular motion of 180° on all three dimensional plains (x,y,z axes) as depicted in figures 16 and 17. The value leap at the beginning of the diagram in the gyroscope charts is caused by the activation of the sensors. It has no meaning for the actual measurement.

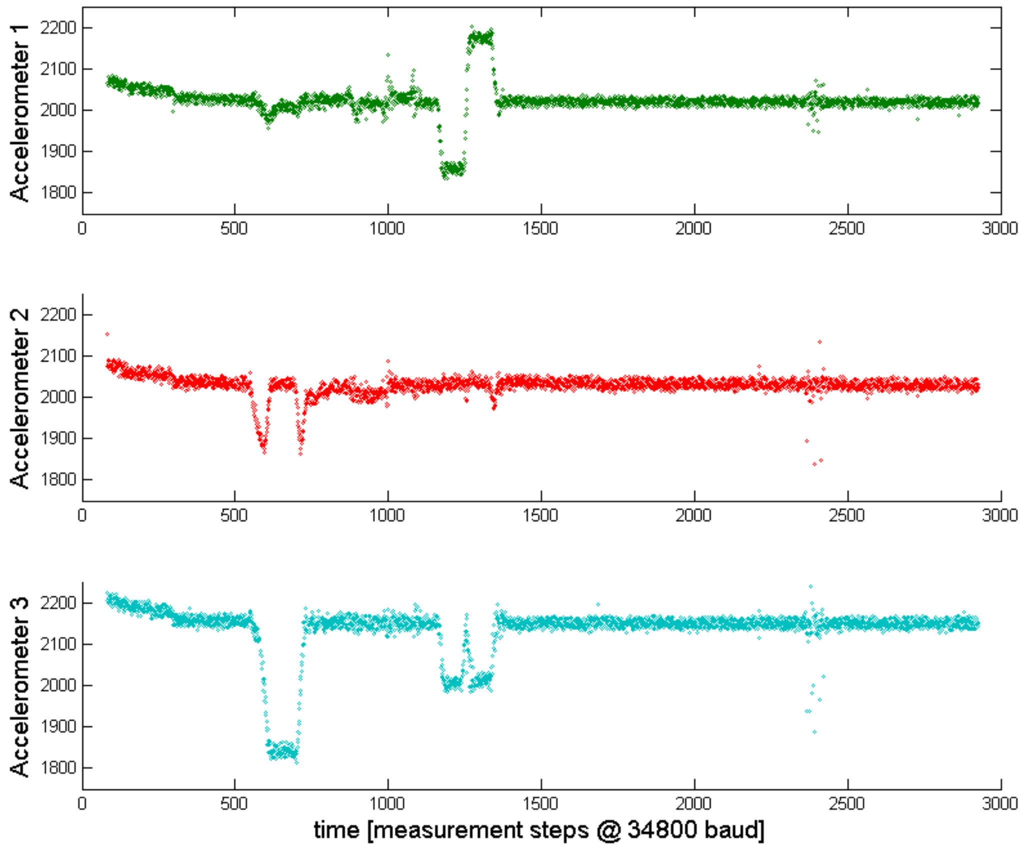


Figure 16: Accelerometers

[30] A **gyroscope** is a device for measuring or maintaining orientation, based on the principle of conservation of angular momentum. In physics the angular momentum of an object with respect to a reference point is a measure for the extent to which, and the direction in which, the object rotates about the reference point. http://en.wikipedia.org/wiki/Gyroscope (Accessed December 2005)

[31] An **accelerometer** is a device for measuring acceleration. An accelerometer inherently measures its own motion. http://en.wikipedia.org/wiki/Accelerometer (Accessed December 2005)

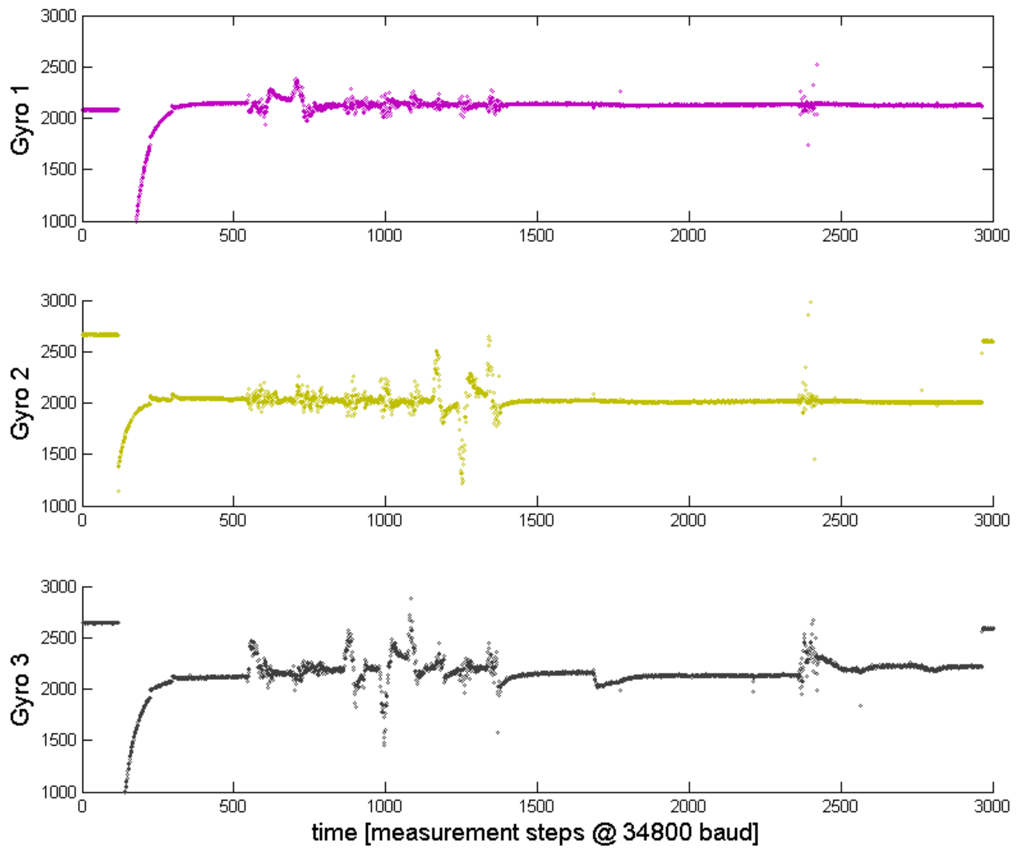


Figure 17: Gyroscopes

4.4 Battery consumption

An important factor when conducting experiments on mobile devices is battery consumption. All the more if the smartphone additionally has to provide power to external devices such as, in our case, the sensorboard. On standby, Palm Inc. guarantees a standard battery life of 300 hours for the Treo 650 smartphone. During continuous talking, the battery life shrinks to 5 hours. In order to find out the battery consumption of the various streams, measurements have been made in an interval of 30 minutes each, for 9 hours straight.

The following graph shows the battery consumption over time with three different configuration settings. The ***all streams***-setting activates, as the name suggests, all available streams (sensorboard-serial / GPS / audio / data storage onto SD Card / rest). The ***sensorboard only*** and ***sound only***-settings should be self-explanatory. The battery consumption of the GPS and the residual internal sensors is practically inconsequential. All measurements pertaining the sensorboard were done with switched off gas sensors.

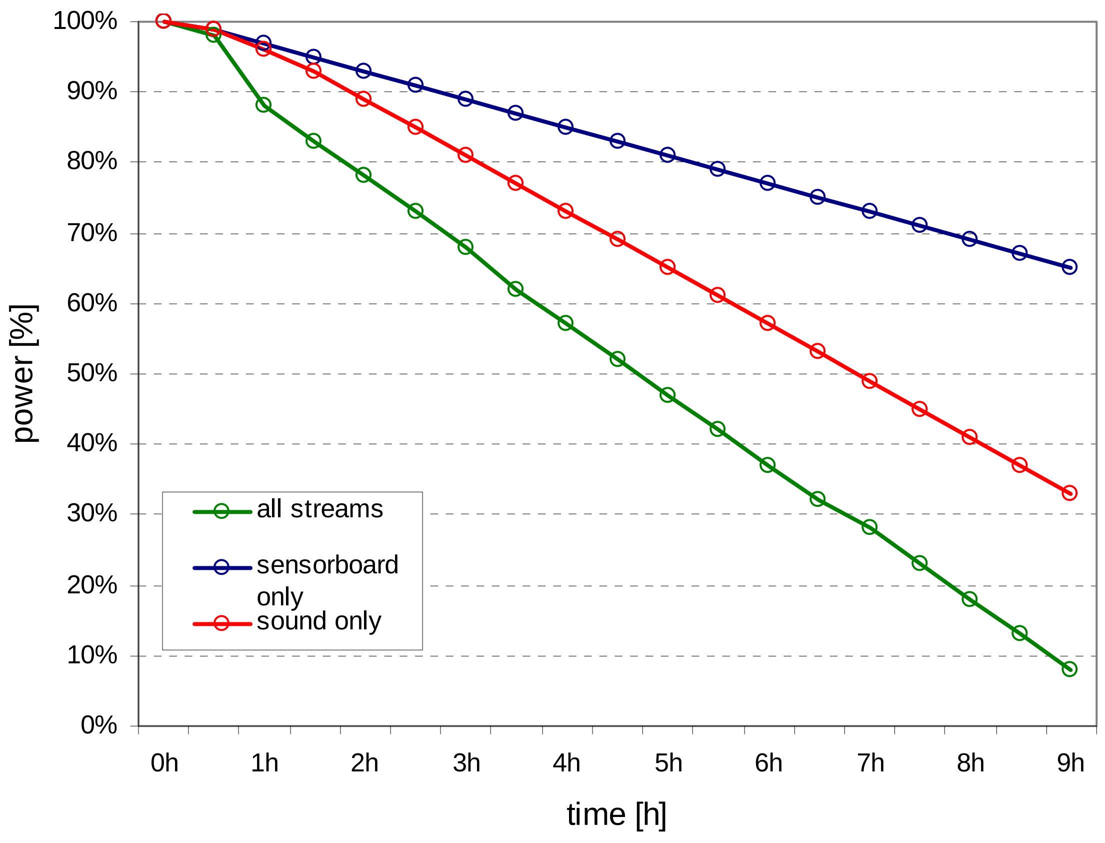


Figure 18: Battery Consumption

Starting from a fully charged battery, a recording session with all data streams turned on, lasts about 9 hours. The sensor consuming the most power turns out to be the audio recorder with an average battery drain of 4% per half an hour, followed by the sensorboard with a constant 2% per half an hour. There is no difference in power consumption whether the sensors on the board are turned on or off, which in this case means that the smartphone itself is the biggest battery consumer. Table 4 illustrates the various power usages in *mAh*.

	power usage
sensors switched off	~50mA
sensors switched on (no gas)	~55mA
sensors switched on (with 1 gas sensor)	~150mA
sensors switched on (with 2 gas sensor)	~250mA

Table 4: Power usage in mAh

The battery pack of the Treo 650 has a charge of 1800*mAh*, which over a period of 9 hours corresponds to a usage of about 200*mAh*. If additionally to the already running sensors, both gas sensors get activated, the battery life of the Treo last half as long, that is approximately 4 hours. In order to prolong the battery life, the gas sensors can be switched on sporadically and for short periods of time[32].

[32] Approximately 1 minute, because it takes about 20-30 seconds to heat up the sensor in order to be ready for reliable measurements.

Chapter 5

Conclusions & Future Work

Chapter 2 has specified the requirements needed in order to implement an application capable of sensing context. Most of those requirements have successfully been met. A few, while definitely feasible, still need to be implemented. Others can not be implemented in compliance with the requirements, due to either hardware or software constrictions. In such cases, unorthodox methods in form of workarounds have to be adopted wherever possible.

The following listing gives an overview of past achievements, current work in progress and future workarounds.

Already implemented and working functionality

- *Framework for session recording*
  The `ContextRecorder` application developed by [Fornallaz 04] and implemented by [de Simoni 05] offers a solid framework in which to embed internal as well as external sensor streams.

- *Sensorboard integration*
  The sensorboard successfully sends data at different baud rates, which the Treo 650 receives and stores.

- *GPS integration*
  The smartphone receives the GPS data stream by the use of the wireless Bluetooth protocol and also stores it.

- *Audio recording*
  Audio streams can be recorded with various degrees of quality depending on parameter settings such as bit depth, sample rate and the number of channels used. The recording thereafter gets stored in a wav-compatible format.

- *Integration of various smartphone-internal sensors*
  By accessing Treo specific phone libraries, several sensors could additionally be integrated.

Future work

- *Phone event interception*
  The theoretical API structure to implement this functionality is available. However, only empirical tests will show if the API truly allows to completely intercept an incoming phone call.

- *Multithreading and background processing*
  As stated earlier, the current Palm OS version on which the Treo 650 runs, does neither support multithreading nor background processing. The workaround proposal described in Appendix C could be able to bypass this restriction.

- *Persistent storage of image data*
  Currently it is possible to employ the camera API to take pictures and display them on the screen, but not to persistently store them. It could be feasible to circumvent this constraint by means of a workaround.

Personal assessment

In order to enable the smartphone to be aware of context, it has to be capable of gathering as much data about its surroundings as possible. Sensors, external as well as internal, provide this opportunity. The more sensors the application is able to integrate, the better are the chances of finding relevant data. So far we have been able to successfully integrate the majority of sensors at our disposal, with the exception of the camera on the smartphone[33].

Another important aspect deciding on the success of the project encompasses the background processing, that is the multithreading capability of the application. Even though multithreading is not supported by the current operating system, the framework of event notifications described in Appendix C has a sound probability to circumvent this barrier.

In case of a successful interception of incoming phone calls[34], the project should have an excellent prospect of success.

[33] The image capturing application has for the most part already been implemented. The only missing functionality consists in storing the captured image onto the expansion card, a problem that could be solvable by bypassing the problematic API functions.
[34] The API to implement this functionality are readily available according to the Palm Inc. API reference.

Appendix A

The Treo 650 smartphone

specifications
wireless radio • GSM/GPRS model: 850/900/1800/1900 MHz world phone
processor • Intel™ PXA270 312 MHz processor
memory • 23MB user-available stored non-volatile memory (22MB multi-lingual)
battery • Removable rechargeable lithium ion battery • GSM/GPRS model: Up to 6 hours talk time and up to 300 hours standby time
operating system • Palm OS® 5.4
weight • 178 grams
display • Colour TFT touch-screen • 320 x 320 resolution • 16-bit colour (displays over 65,000 colours)
expansion • Supports SD, SDIO and MultiMediaCards
• Bluetooth wireless technology

The test device on which the data gathering application runs is the Treo 650 smartphone from Palm Inc. It has built-in audio and video recording capabilities, is equipped with an UART interface ideal for receiving data from our sensorboard and features a quite extensive API support, thus allowing for a high flexibility concerning the programming. The Treo 650 is a pen based smartphone running on the Palm operating system v5.4[35].

[35] For further information concerning the Treo 650 visit http://www.palm.com. To acquire additional information regarding the OS, visit http://www.palmsource.com. (Accessed December 2005)

Appendix B

The Socket GPS receiver

specifications
size • 50 x 84 x 20 mm
power • Removable rechargeable lithium ion battery with 5V DC input charging circuit (650 mA)
operation time • Default: 9 hours after full charge, continuous mode, 25°C
accuracy • Position: 10m, RMS, 25m CEP without SA • Velocity: 0.1 m/sec without SA • Time: 1 microsecond synchronized to GPS time
channels • 16 Channels all-in-view tracking
antenna type • Built-in Ceramic patch antenna
dynamic conditions • Altitude: < 18,000 m • Velocity: < 515 m/sec • Acceleration: < 4 g

The GPS receiver provides its data via the Bluetooth protocol directly to the Treo 650 smartphone. The data stream is NMEA-0183 (v2.30) compatible, which allows to add GPS position technology to any application capable of parsing that standard. Thanks to the wireless transfer method, the device can unobtrusively be stashed into a jacket pocket for example, while conducting outdoor measurements[36].

[36] Consult the manufacturer's website for further information regarding the Socket GPS receiver. http://www.socketcom.com/product/GP0820-521.asp (Accessed January 2006)

Appendix C

Event notification framework

As described in the implementation section, the operating system currently running on the Treo 650, does not support multithreading. Due to the fact that the running application quits as soon as a new one gets started, it becomes difficult to implement the data gathering application as a background process. To be able to obtain meaningful data none the less, we need to construct a workaround capable of influencing the event-driven operation method of the OS itself[37].

The key to an event-driven notification framework lies in the ability to launch Palm OS applications based on certain events occurring on the system[38]. An application launches when it receives a launch code. Upon receiving the launch code, the application initializes itself, it then goes into an event loop, and exits by means of a deinitialization. The default launch code is `sysAppLaunchCmdNormalLaunch`, allowing the application to start in the foreground with a full functioning GUI. There is a whole array of customized launch codes ready to use[39]. The two key functions allowing to launch an application from inside another one, and most importantly, to restart the previous application after the invoked one has quit, are the `SysAppLaunch` function and the `SysUIAppSwitch` function subsequently delineated.

[37] This framework is mostly a theoretical construct. While most key features should be viable to implement, the framework has to be tested as a whole subsequently.
[38] A Palm OS application does launch when the user requests it, but it may also launch in response to some other user action, such as a request for the global find facility.
[39] For a complete listing of available launch codes consult http://www.palmos.com/dev/support/docs/palmos/PalmOSCompanion/AppStartupAndStop.html#998407 (Accessed December 05)

C.1 Launch functions

SysAppLaunch Function	
Purpose	Launch a specified application as a subroutine of the caller
Declared In	`SystemMgr.h`

Prototype	`Err SysAppLaunch (`
	`   UInt16 cardNo,`
	`   LocalID dbID,`
	`   UInt16 launchFlags,`
	`   UInt16 cmd,`
	`   MemPtr cmdPBP,`
	`   UInt32 *resultP`
	`)`

Parameters	→ *cardNo, dbID* The card number and ID of the resource database of the application to launch. → *launchFlags* Set to 0. → *cmd* Launch code. → *cmdPBP* Launch code parameter block. ← *resultP* The value returned from the application's `PilotMain()` routine.

Table 5: SysAppLaunch Function

`SysAppLaunch` can be used to send specific launch codes to other applications and have control return to the calling application when finished. Making the invoked application a subroutine of the caller has its disadvantages though. For instance no global variables can be accessed. In addition, if the application has multiple code segments, no code outside of the first segment can be accessed, therefore diminishing our options considerably.

A slightly different and more promising approach is offered by `SysUIAppSwitch`. This function tries to make the current application quit, subsequently invoking the application specified by the card number and database ID with full privileges and hence access to global variables[40].

[40] The only way to launch an application without restrictions can be achieved by using the `sysAppLaunchCmdNormalLaunch` code utilized by `SysUIAppSwitch`.

SysUIAppSwitch Function	
Purpose	Try to make the current UI application quit and then launch the UI application specified by card number and database ID.
Declared In	`SystemMgr.h`

Prototype	`Err SysUIAppSwitch (`
	`    UInt16 cardNo,`
	`    LocalID dbID,`
	`    UInt16 cmd,`
	`    MemPtr cmdPBP`
	`)`

Parameters	→ *cardNo* Card number for the new application; currently only card 0 is valid. → *dbID* ID of the new application's resource database. → *cmd* Action code (launch code). → *cmdPBP* Action code (launch code) parameter block.

Table 6: SysUIAppSwitch Function

The code example shows how to programmatically launch an application, let's call it the TargetApp, from our own application.

```
static void MainGotoApplication()
{
  LocalID theDBID;
  UInt theCardNo;
  DmSearchStateType theSearchState;

// Grab the id of the application we want to launch (the TargetApp)
  DmGetNextDatabaseByTypeCreator(true, &theSearchState,
sysFileTApplication, sysFileTargetApp, true, &theCardNo, &theDBID);

// Launch the new app.
  SysUIAppSwitch(theCardNo, theDBID, sysAppLaunchCmdNormalLaunch,
NULL);
}
```

C.2 Notifications

The launch code driven approach has one major flaw that needs to be eliminated. `SysAppLaunch` allows us to invoke subroutines of other applications limiting the operations available. At the same time our application always stays in the foreground, thus making a reasonable user interaction with the device's other applications practically impossible. `SysUIAppSwitch` on the other hand exits the current application, starts the target application, and if desired, restarts the invoking application. The problem is the same as in `SysAppLaunch`, namely the starting point. Both functions have to be invoked from our own application, forcing to start our application manually after each interruption [41].

Notifications complement the framework by patching aforementioned flaw. Applications can receive notifications and launch when certain system-level events or application-level events occur. Notifications are similar to application launch codes, but differ from them in two important ways:

1. Notifications can be sent to any code resource, such as a shared library or a system extension. Launch codes can only be sent to applications. Any code resource that is registered to receive a notification is called a notification client.
2. Notifications are only sent to applications or code resources that have specifically registered to receive them, making them more efficient than launch codes. Many launch codes are sent to all installed applications to give each application a chance to respond[42].

Applications have the choice to register only for needed notifications. This approach lowers the communication overhead allowing for a more flexible configuration depending on the circumstances. Table 7 enumerates a selection of available notifications[43]. In our case, we always want to be notified of system as well as application level events, especially when our application is not running. For that purpose we need to register our notification listeners upon receiving the `sysAppLaunchCmdSystemReset` launch code. This procedure guarantees our application to be notified of arbitrary events even when not running or after system resets.

[41] After a system restart, caused by a crash or a dead battery for example, the user explicitly has to restart our application. Due to forgetfulness or lack of knowledge, this situation could easily lead to a prolonged shutdown of our data gathering application without the user noticing.
[42] http://www.palmos.com/dev/support/docs/palmos/PalmOSCompanion/AppStartupAndStop.html#998657 (Accessed December 05)
[43] For a full listing of available notifications consult the Palm OS Programmer's API Reference.

Constant	Description
`kTelTelephonyNotification`	**A telephony event has occurred.**
`sysExternalConnectorAttachEvent`	A device has been attached to an external connector.
`sysExternalConnectorDetachEvent`	A device has been detached from an external connector.
`sysNotifyAppLaunchingEvent`	**An application is about to be launched.**
`sysNotifyAppQuittingEvent`	**An application has just quit.**
`sysNotifyCardInsertedEvent`	An expansion card has been inserted into the expansion slot.
`sysNotifyCardRemovedEvent`	An expansion card has been removed from the expansion slot.
`sysNotifyDeviceUnlocked`	The user has unlocked the device.
`sysNotifyEarlyWakeupEvent`	The system is starting to wake up.
`sysNotifyEventDequeuedEvent`	An event has been removed from the event queue with `EvtGetEvent`.
`sysNotifyGotUsersAttention`	**The Attention Manager has informed the user of an event.**
`sysNotifyHelperEvent`	An application has requested that a particular service be performed.
`sysNotifyIdleTimeEvent`	The system is idle and is about to doze.
`sysNotifyLateWakeupEvent`	The system has finished waking up.
`sysNotifyNetLibIFMediaEvent`	The system has been connected to or disconnected from the network.
`sysNotifyResetFinishedEvent`	**The system has finished a reset.**
`sysNotifySleepNotifyEvent`	The system is about to go to sleep.
`sysNotifySleepRequestEvent`	The system has decided to go to sleep.
`sysNotifyVolumeMountedEvent`	A file system has been mounted.
`sysNotifyVolumeUnmountedEvent`	A file system has been unmounted.

Table 7: Notification Constants

Appendix D

Sensorboard data stream

The sensorboard sends a continuous data stream through the serial connection without flow control mechanisms to regulate its output. It is therefore crucial that the receiving end of the stream, in our case the incoming serial buffer of the smartphone, is capable of correctly receiving and storing the incoming data. As described earlier in the generic Palm OS architecture section of chapter 3, the event manager is responsible for the periodic checking of the event queue and the ensuing handling of possible event occurrences. The frequency with which the event manager checks for new events can be indicated in system ticks and is illustrated in following code extract[44].

```
void AppEventLoop( void ) {
  do {
    //The second parameter of the EvtGetEvent function defines the
    //amount of system tick per scan cycle
    EvtGetEvent( &event, 1 );
    if (SysHandleEvent( &event ) == false) {
      if (MenuHandleEvent( 0, &event, &error ) == false) {
        if (AppHandleEvent( &event ) == false) {
          FrmDispatchEvent( &event );
        }
      }
    }
  } while (event.eType != appStopEvent);
}
```

Table 8: The application event loop

Since the size of the receiving buffer consists of a fixed value, the only two parameters in a position to influence the data stream are composed of the sensorboard baud rate and the system tick parameter. Experiments with various baud rates and system tick settings have delivered inconclusive results. The receiving accuracy of the raw sensorboard values is difficult to predict and control. Hence there could arise the need for a parser in the future, which, located between the receiving buffer and the storage routine, would be capable to

[44] A system tick is a Palm device internal time measurement unit. The Treo 650 apparently has a value of approximately 100 system ticks per seconds (although this number can vary up to ± 0.9 ticks per second).

trim the incoming data stream regardless of baud rate, but accepting a minor loss in data throughput due to.trimming attrition.

Table 9 shows a series of measurements on raw sensorboard values as they are stored in a text file on the expansion card. The samples are ordered by baud rate. The recording per sample lasted exactly 10 minutes.

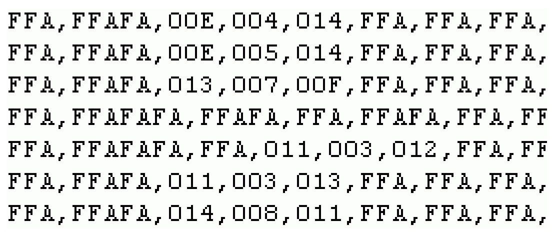
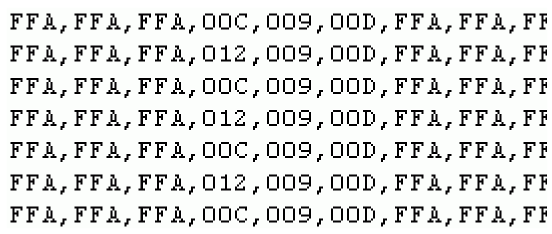
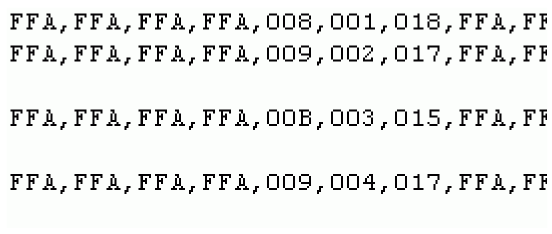
baud rate	file size (kB)	sensor output
9600	112	 FFA,FFAFA,00E,004,014,FFA,FFA,FFA, FFA,FFAFA,00E,005,014,FFA,FFA,FFA, FFA,FFAFA,013,007,00F,FFA,FFA,FFA, FFA,FFAFAFA,FFAFA,FFA,FFAFA,FFA,FF FFA,FFAFAFA,FFA,011,003,012,FFA,FF FFA,FFAFA,011,003,013,FFA,FFA,FFA, FFA,FFAFA,014,008,011,FFA,FFA,FFA,
38400	982	 FFA,FFA,FFA,00C,009,00D,FFA,FFA,FF FFA,FFA,FFA,012,009,00D,FFA,FFA,FF FFA,FFA,FFA,00C,009,00D,FFA,FFA,FF FFA,FFA,FFA,012,009,00D,FFA,FFA,FF FFA,FFA,FFA,00C,009,00D,FFA,FFA,FF FFA,FFA,FFA,012,009,00D,FFA,FFA,FF FFA,FFA,FFA,00C,009,00D,FFA,FFA,FF
115200	321	 FFA,FFA,FFA,FFA,008,001,018,FFA,FF FFA,FFA,FFA,FFA,009,002,017,FFA,FF FFA,FFA,FFA,FFA,00B,003,015,FFA,FF FFA,FFA,FFA,FFA,009,004,017,FFA,FF

Table 9: Sensorboard data stream given various baud rates

As can be seen by the alignment of the values in the sensor output screenshots on the right side of the table and by the size of the recording samples, there seems to be no correlation between the baud rate and the quality of the stored data at a constant system tick. The currently most promising baud rate seems to be at 34800 bits per second. It shows an organized data structure as well as the highest data throughput of all the samples.

List of Figures

List of Tables

Bibliography

[de Simoni 05]	Frederic de Simoni. Auf dem weg zum wirklich smarten smartphone: Anbindung interner und externer sensorik zur kontexterfassung. Diploma thesis in computer science University of Zuurich, 2005
[Dey 01]	Dey A.K. Understanding and using context appeared in: Personal and Ubiquitous Computing, Vol.5
[Donner 05]	Donner Manuel Heterogenität von Benutzergruppen in mobilen, kontextbewussten Anwendungen Diploma thesis in computer science University of Zurich, 2005
[Fornallaz 04]	Fornallaz, J. Fundamental Implementation for Context Sampling on Mobile Phones Diploma thesis in computer science University of Zurich, 2004
[Foster 05]	Lonnon R. Foster and Glenn Bachmann Professional Palm OS Programming Indianapolis, USA, 2005, Wiley Publishing Inc.
[palmOne 04]	palmOne, Inc. Developer Guide 2004, PDF, p.27 http://developers.palm.com/pe/index.jsp
[Roth 02]	Roth, J. Mobile Computing – Grundlagen, Technik, Konzepte. Heidelberg, 2002, dpunkt Verlag

[Siewiorek 03]	Siewiorek, D., Smailagic A., Furukawa, J., Moraveji N., Reiger K., Shaffer J. SenSay: A context-aware mobile phone School of Computer Science Carnegie Mellon University, 2003 http://www-2.cs.cmu.edu/~aura/docdir/sensay_iswc.pdf
[Tschanz 05]	Stephan Tschanz Towards a framework for an inertial navigation system: position and velocity extrapolation of motion-based data Diploma thesis in computer science University Zurich, 2005
[Vorburger 05]	Peter Vorburger, Abraham Bernstein and Alen Zurfluh The Artificial Receptionist: Anticipating a Person's Phone Behavior Department of Informatics, University of Zurich MCMP-05, 2005.
[Zurfluh 04]	Zurfluh, A. The artificial secretary. Diploma thesis in computer science University of Zurich, 2004