3-19-2008

# A Framework for Evaluating the Computational Aspects of Mobile Phones

David Pedro Aguilar
*University of South Florida*

# A Framework for Evaluating the Computational Aspects of Mobile Phones

by

David Pedro Aguilar

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Rafael Perez, Ph.D.
Miguel Labrador, Ph.D.
Dewey Rundus, Ph.D.
Edward Mierzejewski, Ph.D.
Gregory McColm, Ph.D.

Date of Approval:
March 19, 2008

Keywords: Benchmarking, Wireless, Cellphones, Global Positioning System (GPS), Application Development, Application Programming Interface (API), Power Consumption

**Acknowledgments**

I wish to thank Dr. Rafael Perez, my major professor, for his consistent motivation, which was of great service to me in prioritizing my time during the completion of this dissertation. I would also like to express my appreciation to the Center of Urban Transportation Research for access to its equipment and other resources in developing and testing the software required by this research, and in particular Sean Barbeau for his frequent assistance.

I am indebted to the support of my friends and family during the course of completing my degree, and especially my wife for her understanding and patience.

**Note to the Reader**

Reader: The original of this document contains color that is necessary for understanding the data. The original dissertation is on file with the USF library in Tampa, Florida.

## Table of Contents

## List of Tables

## List of Figures

**A Framework for Evaluating the Computational Aspects of Mobile Phones**

**David Pedro Aguilar**

## ABSTRACT

With sales reaching $4.4 billion dollars in the first half of 2006 in the United States alone, and an estimated 80% of the world receiving coverage for their wireless phones in that year, interest in these devices as more than mere communicators has greatly increased. In the mid-to-late 1990s, digital cameras began to be incorporated into cellphones, followed shortly thereafter by Global Positioning System (GPS) hardware allowing location-based services to be offered to customers. Since then the use of mobile phone hardware for non-communication purposes has continued to expand. Some models, such as the Motorola V3M, have been specifically geared toward the storage and display of music and visual media, as well as receiving Internet broadcasts.

It is perhaps surprising, therefore, that relatively little has been done from an academic standpoint to provide a qualitative and comprehensive method of evaluating the performance of mobile phones regarding their ability to function as computing devices. While some manuals do offer comparisons of Application Programming Interfaces (APIs) that aid in the development of cellphone applications, little documentation exists to provide objective measurements of performance parameters.

This dissertation proposes a framework for evaluating the performance of mobile phones from a computational angle, focusing on three criteria: the processing power of

the Central Processing Unit (CPU), data transfer capabilities, and the performance of the phone's GPS functionality for the appropriation of geographic location data.

Power consumption has always been a major source of interest in the study of computer systems, and the limited hardware resources of mobile devices such as laptop computers, Personal Data Assistants (PDAs) and cellular telephones makes this a key concern. The power consumption factors associated with operation are therefore considered alongside the three core criteria being studied in this framework.

In addition to framework design, software tools for the evaluation of cellphones were also developed, and these were applied to a test case of the Sanyo SCP-7050 model. This provides an example of the utility of the framework in evaluating existing phone models and a foundation for the assessment of new models as they are released.

# Chapter 1

# Introduction

## 1.1 Background

With an estimated 90% of the world to be provided with wireless phone coverage by the year 2010 [1], interest in using cellphones for communication as well as other purposes has been dramatically increasing in recent years commensurate with that expectation. The 80s and 90s saw the advent of a number of additional features such as Internet access [2] and cameras [3] being incorporated into the basic model to expand its use. In 2002 Global Positioning System (GPS) technology was introduced to mobile phone models, with accompanying applications for the display of geographical coordinates [4].

In addition to the wide variety of commercially available applications, some companies such as Motorola and Sprint have provided a number of resources for the third-party development of software to be run on compatible phone models [5]. Tools such as the Eclipse software framework and the NetBeans Integrated Development Environment (IDE) include compilers and device emulators that allow both the creation and implementation of applications in specialized software languages (*e.g.*, Java 2 Micro Edition). The result of this is the increasing popularity of mobile phones as computing agents, performing many of the tasks once restricted to personal computers or laptops such as browsing the web, conducting internet banking [6], electronic gaming, access to

online databases for the storage and retrieval of data, and performing a variety of sophisticated mathematical operations.

This increased recognition is not limited merely to the public arena; interest in researching the computational aspects of mobile phones and similar devices such as Personal Data Assistants (PDAs) has also seen a rise in recent years [7]. The small size of the input and output devices, as well as the restricted methods of data entry (*e.g.*, lack of keyboard and mouse) have made user interface design an important subject of study [8]. In addition, while much of the recent research has focused on the communication aspects of mobile computing [9][10], the computing aspect, with related factors such as power consumption [11] and heat dissipation [12], has not entirely escaped notice either.

### 1.2 Motivation

While academic interest in the mobile phone as a computational device has indeed been increasing, little publicly-available information has been released to provide a comprehensive and qualitative understanding of the performance of cellphones in terms of their processing and communication capabilities, and certainly no unifying view of the systems' functionality. Developers of software applications for use with mobile phones are provided with some resources for performance comparisons of the various programming interfaces available for their use [13], but without established benchmarks truly informed decisions are difficult to make.

With more sophisticated applications being developed, particularly those that integrate various aspects of device functionality, an awareness of the capabilities of location-aware mobile computing systems, and the level of performance displayed in the

execution of these functions, is becoming a critical issue. Developers that are unaware of the devices' abilities may be unable to efficiently produce new software, while those who are unfamiliar with tradeoffs of performance may produce applications that are rendered ineffective in practical use due to limiting factors such as battery life, communication restrictions or interface issues.

The motivations driving this dissertation are therefore twofold. First, the existing methods of computer performance evaluation are examined for applicability to cellphones. With the specialized hardware and software being run on mobile devices, not all of the methods currently used for the assessment of computing systems may prove useful. Second, an evaluation framework is developed that attempts to address the problems arising from a lack of performance measures publicly available for developers and researchers of mobile computing and communication technology.

### 1.3 Contributions

This dissertation, in developing and testing an organized method for the evaluation of the computational aspects of cellphones, deals with several inter-connected aspects of the technology. The resulting contributions of this work are primarily these:

- The development of a structured technique (*i.e.*, a framework) designed to apply current and tailor-made evaluation methods to mobile computing systems, resulting in a set of conceptually linked metrics that provide quantitative measures of system abilities. Due consideration is given to the unique factors involved with such mobile systems, including:

- Power consumption as militated by the continual operation of various applications
- The applicability of current methods of computer performance evaluation to mobile systems

- An evaluation report of a specific cellphone model's performance is generated by means of this framework in order to provide a case study demonstrating its use.

## 1.4 The Organization of This Dissertation

The remainder of this dissertation is organized as follows:

- Chapter 2 deals with the history and development of mobile phones, focusing on the hardware, software and interface components that lend themselves to computing tasks.
- Chapter 3 discusses the methods of benchmarking commonly applied to computing devices and examines their applicability to mobile phones.
- Chapter 4 presents the design of the evaluation framework, including the developer environment and application development.
- Chapter 5 describes the application of the evaluation framework to a Sanyo SCP-7050 cellphone, and reports on the performance measures in the areas of CPU processing power, data transfer capabilities and the interaction of applications with embedded GPS hardware. The critical relationship of these operations to battery consumption is also discussed.
- Chapter 6 provides a summary of the framework's use as a means of performance evaluation, and considers the direction of further work in this area.

# Chapter 2

# Basic Concepts

## 2.1 Cellular Phones as Location-Aware Mobile Computing Systems

### 2.1.1 Development of the Current Model

When the concept of mobile telephones was first being discussed in AT&T Bell labs in the 1940s, computer systems, which had extremely limited processing power by today's standards, occupied entire rooms [14]. With the invention of transistors in 1947, the large and fragile vacuum tubes that formed much of the bulk of the computers' hardware began to be replaced by components that were both smaller and more energy efficient. The size reduction and other factors also resulted in a dramatic increase in the computational speed of these devices. The development and improvement of computer hardware has continued, widely studied, to this day [15].

Communication technology, fuelled by the need to exchange data, has likewise developed rapidly in the past few decades, going from simple terminal-to-terminal hookups to Wide Area Networks (WANs) and distributed wireless systems that span the entire globe [16][17].

With the advent of the microcomputer, implementing wireless communication in a hand held device became practical for the first time [14], and as miniaturization has continued, the hardware required has gone from the heavy, fixed car phones of the 1970s,

through the large but portable Motorola DynaTAC [18], to the pocket-sized devices of today.

### 2.1.2 Strengths and Weaknesses of the Mobile Design

The modern cellphone design contains many features that make it convenient for use, such as its size, minimal weight and extensive areas of communication coverage; by the end of 2006 more than 80% of the globe had wireless connectivity [19]. The small size of current processing units and electronic storage media has also vastly increased the devices' computational power, opening up a wide array of features above mere voice transmission.

#### 2.1.2.1 Hardware Issues

The design of cellphone hardware is oriented toward very different objectives than that of other computing devices. Although the basic structure is essentially the same, (Figure 2.1) the design goals are strongly focused on minimizing the size and weight of components. With the benefits of portability, however, come a number of drawbacks not common in stationary devices, and many of the resulting tradeoffs are directly related to this design focus.

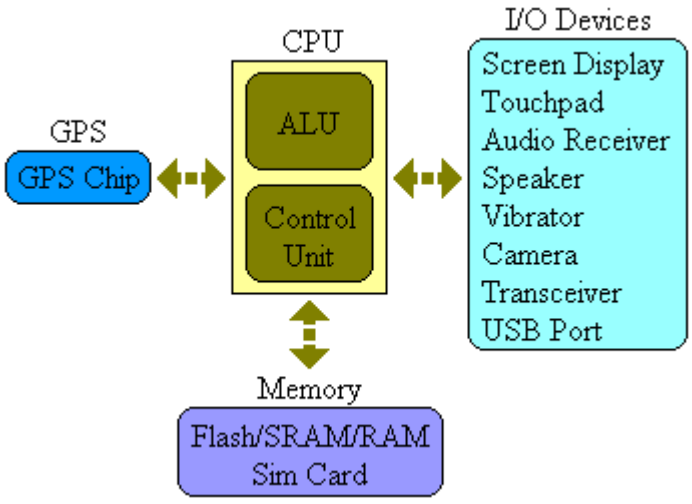


Figure 2.1 – Basic Hardware Architecture

Due to their small size, mobile communication and computational devices such as cellular phones suffer from many of the same issues as do laptop computers, only more acutely. Some of the limitations related to hardware constraints include:

- Memory limits – The standard data storage capacity in a high-end modern cellphone is in the order of one or two gigabytes, even with additional hardware such as micro SD cards [20]. While this may seem like a large amount of space when seen from a communication standpoint, viewing this from a mobile computing perspective, particularly one that may be used for the storage and utilization of multimedia resources, the limitations readily become visible.
- Processor size (and speed) – A few hundred megahertz of processing power characterizes most cellphones today [21]. Compared to modern desktop machines, any intensive computational tasks such as visual face recognition [22] take a significantly longer time.

- Power consumption – Battery life is a major consideration when designing software for cellphones intended to work over an extended period. Since a phone that is not running any applications can only be active for a matter of days, and performing any continuous operation will drastically reduce this time, (see Chapter 4) this presents an obvious disadvantage over stationary systems that remain connected to a continuous power supply.
- Specialized parts – While PCs and workstations are fairly flexible in the components they employ for operation, cellphone manufacturers tend to produce company-specific components that are incompatible with competitors' devices, and often other models produced by the same company. The availability of parts in the event of damage or malfunction is therefore a critical issue. In addition, cellphones are not easily upgraded, as are desktop computers, therefore the limits imposed upon software developers by the current specifications of the mobile phones with which they are working tend to be absolute [23].
- Peripheral support – Data representation and transfer on laptop and desktop systems has become a relatively simple affair. Printers, CDs, diskettes and tapes, which are still used for data backup, are all available options for information storage, transmission and display. Cellphones, however, possess limited resources for displaying information (see Section 2.1.2.2) or outputting data to hardcopy and other forms of long-term storage. If the information maintained in cellphone memory is not first transferred to a system already connected to these output devices, specialized hardware adapters must be employed.

#### 2.1.2.2 User Interface Issues

While cellphones, because of their mobility, do offer some advantages over stationary systems in terms of environmental interaction (*e.g.*, a portable digital camera) the data representation methods employed suffer from some obvious limitations due to screen size. Power consumption is also an important factor for the operation of high-resolution, color displays [24]. Sounds, likewise, are triggered by requests to the device [25], and are often unreliable, which becomes an issue if an application is attempting to use an audible alert for real-time events.

Input devices are another major factor. Desktop computers and laptops utilize keyboards and other peripherals such as mice and trackballs to enable the rapid entry of information from the user. Using cellphones, which rarely support external data entry devices, and which rely upon a numerical keypad for input, generally involves a relatively slow and tedious process, particularly when alphanumeric information is required [26]. While voice activated commands have begun to be incorporated into cellphones for help with dialing numbers, if such technology is extended to enable the facilitation of more complicated tasks, the question of where this voice software would be located (*i.e.*, locally or on a remote server) and other factors come to the fore [26].

#### 2.1.2.3 The Cellphone Software Environment

Due in large part to the hardware issues described above, particularly the limited storage space available, the software that runs on cellphones is quite restricted in size. While today's operating systems such as Windows Vista occupy gigabytes of storage space [27], the operating systems of cellphones must be relatively tiny.

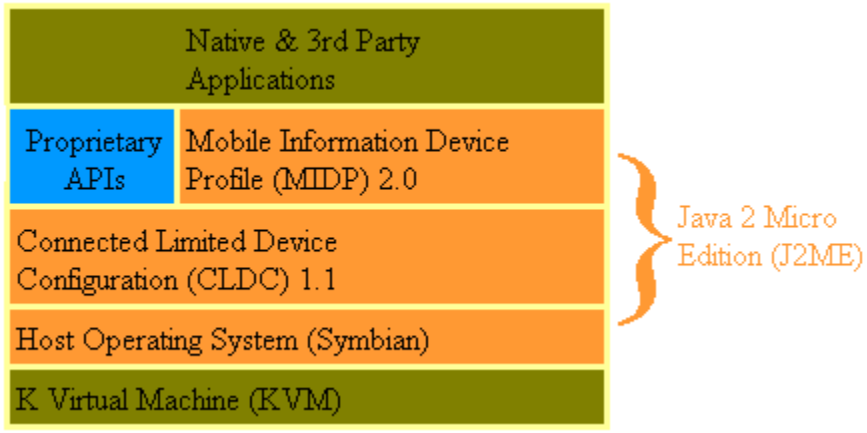


Figure 2.2 – The Cellphone Software Environment

The software applications that run in the software environment (Figure 2.2) must be of a correspondingly small size, and will therefore not have the same capabilities of those implemented on dedicated computing machines [28]. While more recent application development environments have begun to introduce functionality vital for serious computing, (*e.g.*, support for floating point operations) they continue to operate on devices supplying only kilobytes, as opposed to megabytes or gigabytes, of memory.

Developers also need to deal with reduced instruction sets for creating applications, the absence of software libraries that may contain useful tools, and workarounds necessary for the software to function in the limited memory and space provided by the cellphone hardware.

## 2.2 The Communication Framework

Although this dissertation focuses mainly on computational aspects, the primary function of cellphones remains the transmission of voice data between handsets. Understanding the way in which connections are made between cellphones for audio

communication provides a basis for examining their communication methods in general, which constitute a vital element of any mobile computing system.

### 2.2.1 Coverage and Call Handling

The call handling network for mobile phones consists of geographic areas of coverage called "radio cells" [29]. The size of these cells is dependent upon the strength of the base stations' signals and environmental conditions, and the radius of coverage may be anywhere between a hundred meters (in urban areas) and a number of kilometers if there are few obstructions.

Each of the base stations, which house the wireless communication equipment at the center of radio cells, is allocated a maximum number of channels for transmitting and receiving data; this is known as the Cell Allocation (CA). In Frequency-Division Multiple Access (FDMA) networks, the same channels may be re-used if there is no overlap in the coverage areas as illustrated in Figure 2.3.

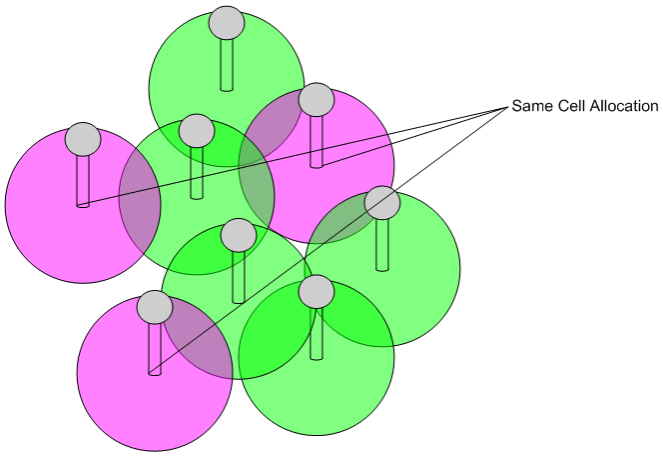


Figure 2.3 – Reusing Cell Allocation Channels

The Global System for Mobile communications (GSM) standard for mobile phones, which is used by most cellphone customers [30], supplies circuit-switched phone services (which consist of dedicated nodes) by means of mobile stations. Valid customers are identified by means of Subscriber Identity Module (SIM) chips that use authentication and encryption keys [31]. The GSM's core network, shown in Figure 2.4, consists of a Switching and Management Subsystem, (SMSS) which incorporates a Mobile Switching Center (MSC), a Gateway MSC (GMSC) and both Home and Visitor Local Registers (HLR & VLR) that store information on subscribers.

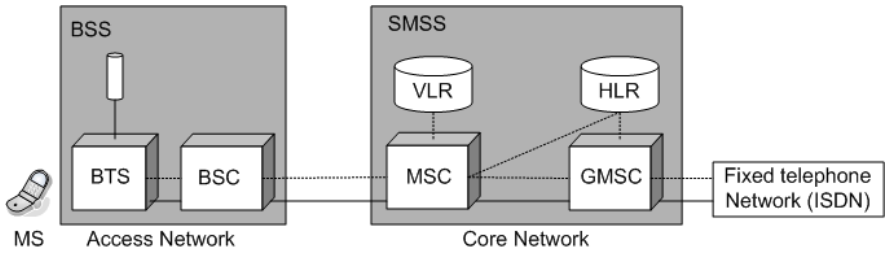


Figure 2.4 – The GSM Core and Access Networks

This core network manages the communication on a high level, coordinating the connection between the mobile phones and the Integrated Services Digital Network (ISDN).

The Access Networks are attached to Base Station Subsystems, (BSS) which each consist of a Base Transceiver Station (BTS) and a Base Station Controller (BSC). The BSS performs, among other functions, the allocation of radio channels to the end-user mobile stations, (the phone hardware) transmission quality management, and the encryption and decryption of the communicated data.

Today's networks support two types of mobility management [32], or the ability of an end-user to communicate dynamically with the network. These are shown in Figure 2.5. In the first management type, Terminal Mobility, an active link is maintained between the network and a user-specific device. The link is dynamic in that the user-device may move in and out of individual coverage areas, and the services are switched over to the appropriate base station in order to maintain connectivity.

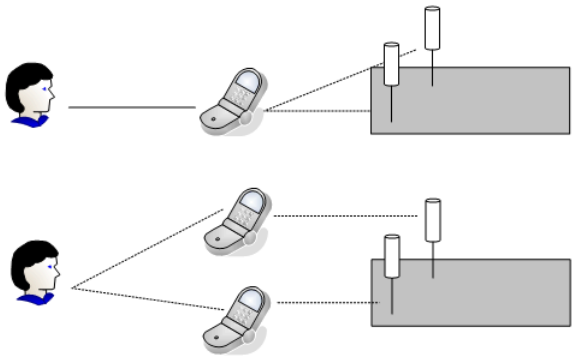

Figure 2.5 – Terminal and Personal Mobility

In the second type, Personal Mobility, the network is able to recognize the subscriber independently of the mobile device used. This is accomplished by means of the unique identifiers such as the International Mobile Subscriber Identity (IMSI). Security issues become important when utilizing the Personal Mobility method, and unlike the user's publicly available telephone number, the IMSI is private in order to protect the customer's identity from misuse.

There are two basic approaches to Location Management, which identifies a user's position with respect to the appropriate Base Station that provides his or her communication services [33]. The first approach is the Location Update method, in which the terminal reports its location to the network at certain intervals, (periodic updates) or

when entering a new radio cell (location-crossing). The second approach is Paging, in which the network searches its cells for terminals, and paged terminals send a response.

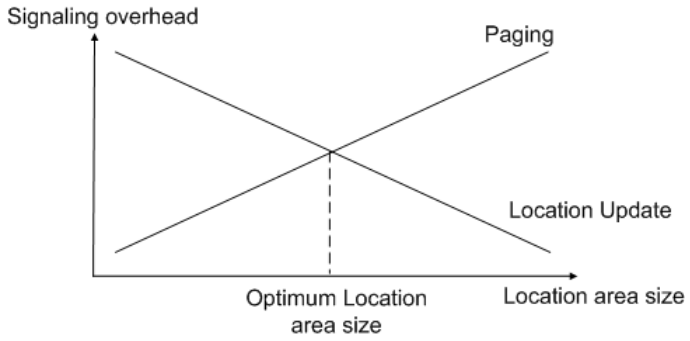


Figure 2.6 – Location Area Size and Location Management

As Figure 2.6 demonstrates, there is a direct relationship between the size of the location area and the transmission overhead associated with the paging and location update methods. Larger location areas mean fewer location updates, but more overhead for the paging method since the latter must search a larger set of potential cells. As it is important, from a quality-of-service and communication cost viewpoint, to minimize the number of signals passed between the network and terminals, choosing the right location area size is key to efficient location management.

### 2.2.2 Data Communication

Mobile devices employ a number of different methods of data communication over the available network architecture. The need for different types of data transmission methods is dictated by the volume and type of data being sent or received. Data that is sent continually and/or has low importance in the overall operation (*e.g.*, periodic GPS updates in the context of a journey being tracked from a remote location) might employ

an asynchronous method such as the User Datagram Protocol (UDP). Data that is vital to the functionality of the application, or of otherwise high importance, may require a more reliable method such as the request/response Hypertext Transfer Protocol (HTTP). The following sections contain brief descriptions of the data communication methods relevant to this dissertation.

#### 2.2.2.1 The Hypertext Transfer Protocol

HTTP stands for "Hypertext Transfer Protocol." It is both generic and stateless; each command is executed independently, without knowledge of any commands that may have come before it. This protocol may be used for transferring data above and beyond the "hypertext" format that its name implies [34]. The method, shown in Figure 2.7, initiates a Transmission Control Protocol (TCP) connection with a port on a remote system, and in return receives a reply that may include a message or an error code.

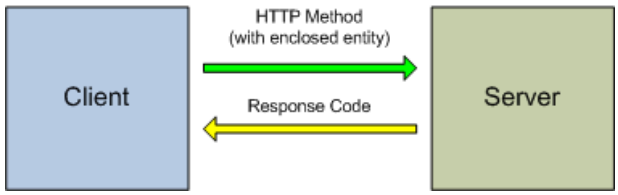


Figure 2.7 – The HTTP Protocol

HTTP defines eight methods [35] used in communication:

- Get – Retrieves the data defined in the Universal Resource Identifier (URI) request.

- Head – Acts as a Get command, but only returns meta-information about the entity described in the URI (*e.g.*, accessibility and recent modification flags).
- Post – Requests that the remote server accept the entity enclosed in the body of the request, identifying the resource to be used in handling the entity.
- Put – Requests that the remote server store the enclosed entity.
- Delete – Requests that the remote server delete the resource identified by the Request-URI (Note: the client system is not guaranteed that this operation is completed, even if the status code returned from the server indicates success).
- Trace – Invokes a remote, application-layer echo of the request message for viewing intermediate servers.
- Options – Returns the HTTP methods supported by the remote server (can be used to check the functionality of a server).
- Connect – A keyword reserved for use with a proxy that can change to being a secure socket tunnel.

HTTP connections may be used in conjunction with Java Servlets in order to communicate with remote servers. Servlets are objects that run within servers in order to receive requests and generate corresponding outputs [36]. These programs can be used to modify information accessible to the server, such as databases, during the course of the web session.

#### 2.2.2.2 The User Datagram Protocol

The User Datagram Protocol (UDP), also known as the Unreliable Datagram Protocol, is used to send packets known as “datagrams” between systems on a network. Figure 2.8 depicts one such datagram packet.

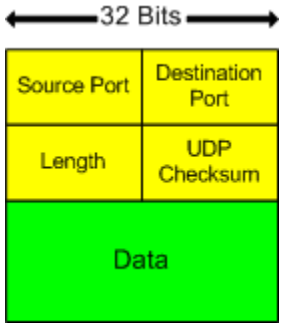


Figure 2.8 – A UDP Datagram Packet

UDP takes messages from the transmitting system, attaches source and destination port number fields, and the resulting segment, as illustrated by Figure 2.9, is then sent to the network layer for transfer over the data link [37].

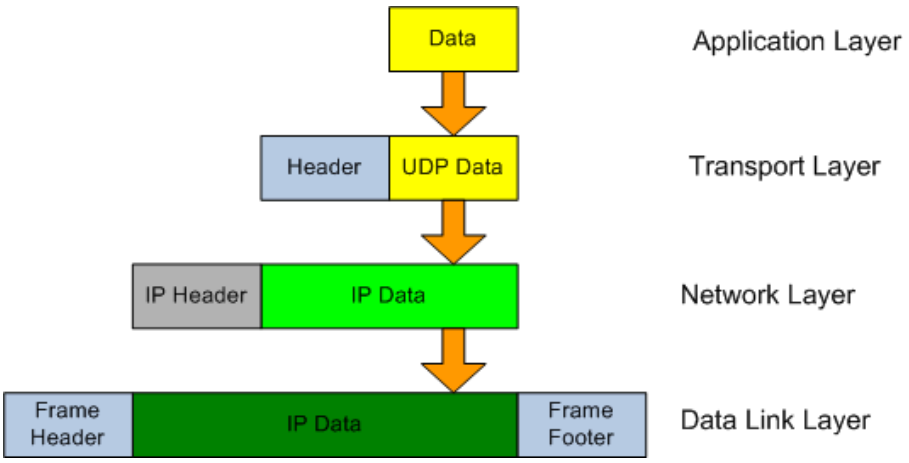


Figure 2.9 – The UDP Communication Protocol Stack

The UDP method provides a number of advantages over response-based protocols, including:

- A larger number of clients – Due to the stateless nature of the transmissions (*i.e.*, each transmission is treated as an independent event), a larger number of clients may be feasibly supported per server, and startup latency is lower.
- Smaller overhead – Connection-based protocols have large header overheads; TCP uses 20 bytes, while UDP uses only 8.
- An unregulated transmission rate – Since transmission congestion controls are not included, the rate at which data can be sent is limited only by the rate at which new data is generated, the capabilities of the transmission device (*e.g.*, processing speed and clock rate) and network bandwidth.

UDP transmissions are, however, unreliable, and the client is not notified regarding the success or failure of its data transmissions. Because of this, not every application or function can make use of it.

### 2.2.2.3 The Transmission Control and Internet Protocols

The Transmission Control Protocol (TCP) is a core protocol for the transmission of electronic information. It is intended to be a very reliable means of data transfer between networked systems [38], and this goal is pursued by means of the assignment of a sequence number to each unit of data transmitted, (see Figure 2.10) and the requirement of a positive acknowledgement signal (ACK) from the receiving network node. The receiving nodes in TCP connections participate in regulating the rate of data flow by transmitting an acceptable range of sequence numbers to the sender.

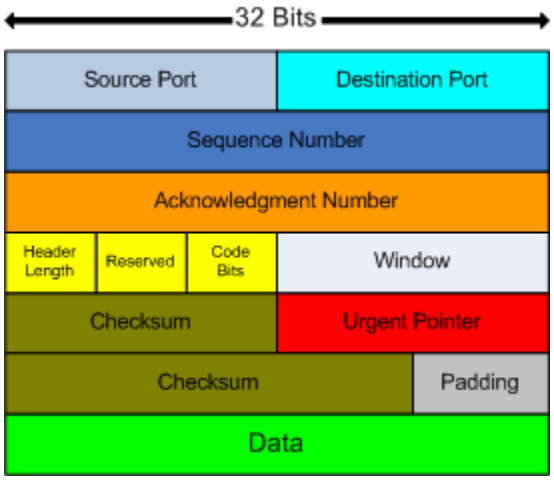


Figure 2.10 – The TCP Segment

The formatting process for data in the TCP method is identical to that of the User Datagram Protocol, (see Figure 2.9) but there are differences in the way these protocols are used, as demonstrated in Figure 2.11, due to key features that set TCP apart from UDP, including:

- Ordered data transfer

- Retransmission of lost packets
- Discarding duplicate packets
- Error-free data transfer
- Flow and congestion control.

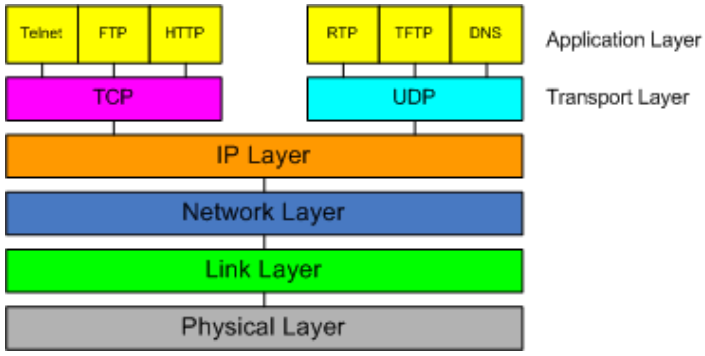


Figure 2.11 – TCP vs. UDP

The TCP connection goes through a number of states [39] described as:

- LISTEN: waiting for a connection request from any remote TCP and port
- SYN-SENT: waiting for a matching connection request after sending a connection request
- SYN-RECEIVED: waiting for a confirming connection request acknowledgment after sending and receiving a connection request
- ESTABLISHED: An open connection for data transfer
- FIN-WAIT-1: waiting for a connection termination request (FIN) from the remote TCP, or the acknowledgment of a connection termination request previously sent
- FIN-WAIT-2: waiting for a connection termination request from the remote TCP

- CLOSE-WAIT: waiting for a connection termination request from the local user
- CLOSING: waiting for a connection termination request acknowledgment from the remote TCP
- LAST-ACK: waiting for an acknowledgment of connection termination request previously sent to remote TCP (includes acknowledgment of termination request)
- TIME-WAIT: delay for enough (twice the Maximum Segment Life, or MSL) time to pass to ensure acknowledgment of its connection termination request by remote TCP
- CLOSED: a "pseudo-state" representing no connection.

The disconnect sequence may be summarized in the sequence [40] depicted in Figure 2.12:

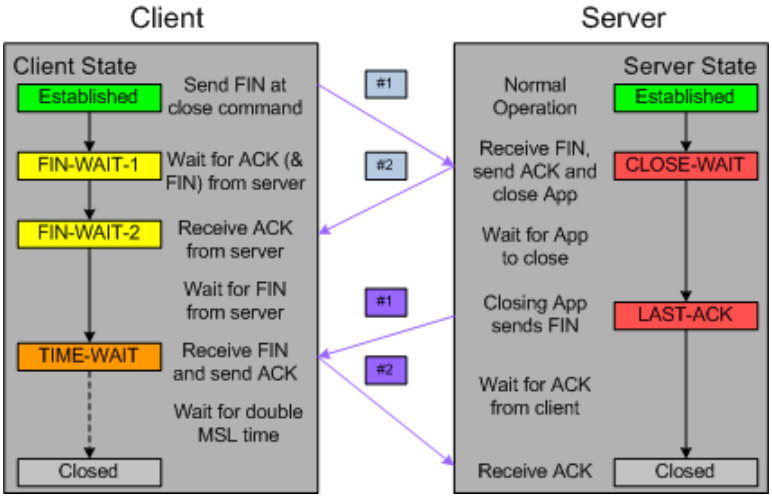


Figure 2.12 – The TCP Connection Termination Sequence

#### 2.2.2.4 The JAX-RPC Protocol

The Java API for XML Based Remote Procedure Calls (JAX-RPC) is based upon the Simple Object Access Protocol (SOAP), which in turn utilizes HTTP for message transfer. Figure 2.13 illustrates the basic functionality of this protocol.

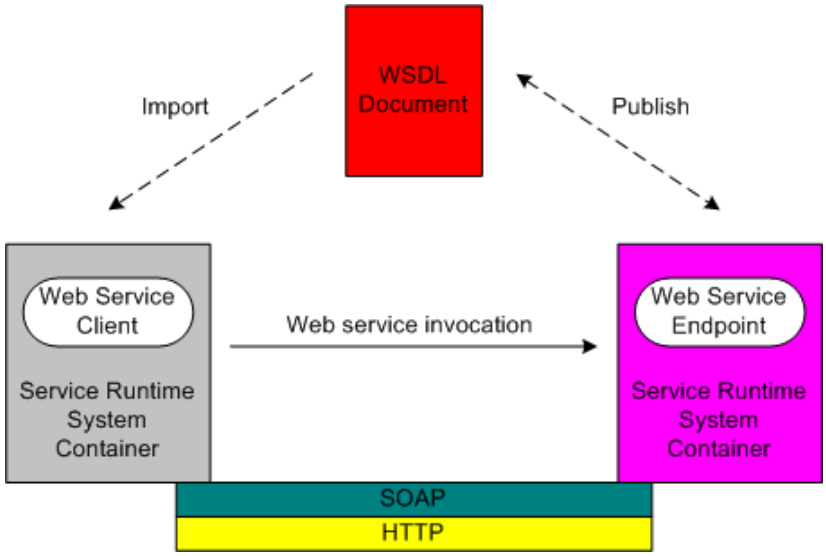


Figure 2.13 – The JAX-RPC Protocol

When used in conjunction with a server-side web service, a Web Service Description Language (WSDL) document specifies a format (in XML) for a service in terms of the endpoints operating on messages [41]. Communication via the RPC method involves two software entities called "Stubs" and "Ties" [42], as shown in Figure 2.14. Stubs are client-side objects consisting of methods that represent procedures on the remote server. The application calls these methods in order to initiate a transfer of information. Ties are classes on the server that provide an interface between the JAX-RPC runtime and the web service.

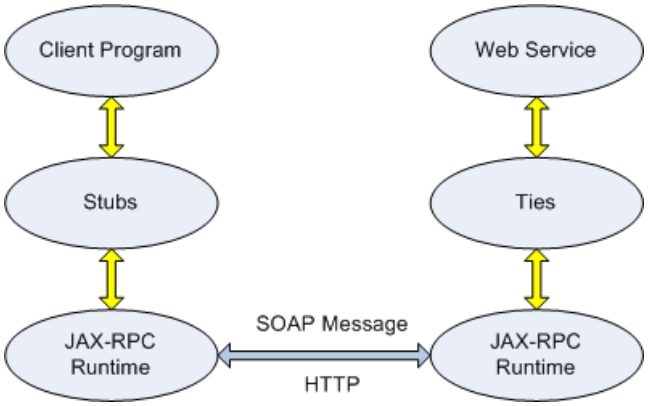


Figure 2.14 – Stubs and Ties

JAX-RPC operates according to the following seven stages:

- The client calls the appropriate method on the stub that represents the remote procedure.
- The stub in turn executes routines on the JAX-RPC runtime system.
- The runtime system converts the routines into a SOAP message for transmission to the server as an HTTP request.
- The server invokes the methods on the server-side JAX-RPC runtime. The JAX-RPC runtime converts the SOAP request into a method call.
- The JAX-RPC runtime calls the method on the appropriate tie.
- The tie object calls the method on the implementation of the Web service.
- An HTTP response is sent via SOAP to the client.

#### 2.2.2.5 Web Services

Web services are server-side software systems that may be used in conjunction with UDP and JAX-RPC transmission methods to perform remote operations in a manner similar to the role that is played by Servlets in strictly HTTP-based applications.


Figure 2.15 – The XML-RPC Protocol Using Web Services

Compatibility with the remote procedure call systems (see Figure 2.15) is due to the SOAP and WSDL specifications at the core of these services, and they provide a standardized method for interoperations between different software applications running on various platforms [43].

### 2.2.3 Location-based Data and Applications

The Global Positioning System (GPS) operates based upon an array of 24 satellites in six orbital patterns above the earth. These satellites broadcast microwave data that may be received by devices equipped with the appropriate hardware. Based upon the signals coming from these sources, a process known as “circular lateration” (see Figure 2.16) is applied, and since the time between the transmission and reception is known, the location of the receiving device can be thereby calculated.

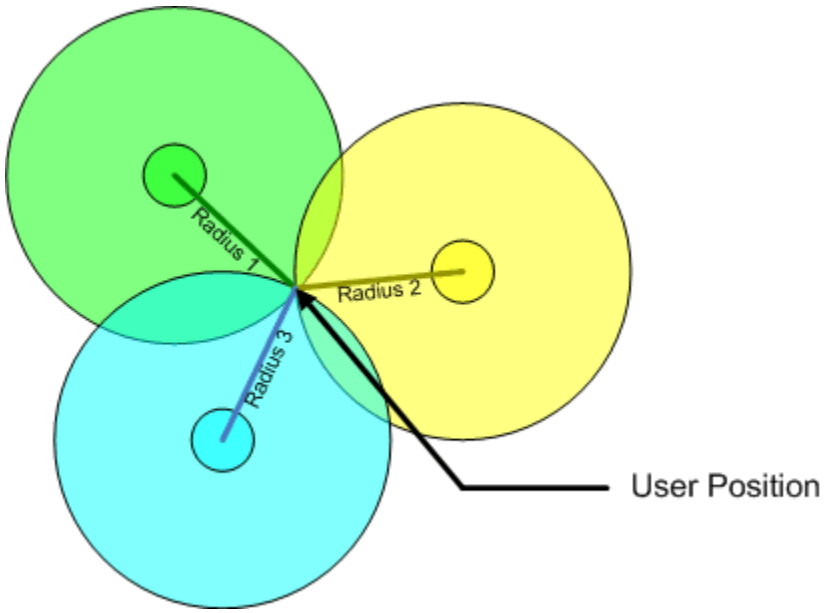


Figure 2.16 – Circular Lateration

Since the incorporation of GPS hardware into mobile phones in 2002 [4], the number and variety of location-based services has diversified greatly. A key example of these services is the e911 system, which enables location information to be sent with emergency calls from cellphones. This quickly alerts dispatchers to the position from which the call is being made [44]. A "reverse 911" component for data-enabled cellphones in the proposed WI-VIA emergency communication system also allows emergency dispatchers to send alerts and warnings to individual within specific geographic areas as selected from a software-generated map [45]. The e911 system has been instrumental in the introduction of positioning technology to wireless cellular networks [44], due to a mandate by the Federal Communications Commission (FCC) [46] that cellular carriers provide user position information with 911 calls that is accurate "to 50 meters for 67% of their calls and to 150 meters for 95% of their calls" [47].

Subsequently developed location-based applications that are available to cellphone customers include:

- eMbience: A MapQuest Traffic and MapApp wireless application [48]
- TomTom: An all-in-one navigation system that includes street navigation, text-to-speech, and real-time traffic information [48]
- Vettro: A system to streamline shipping operations by coordinating GPS location tracking with real-time driver status [48]
- LBSNow: An application that shows users their proximity to particular businesses or services [49]
- Beacon: An ESRI-powered social networking application [49]
- SpotKast: A location-based news and update application [49].

# Chapter 3

# Benchmarking

## 3.1 The Benchmarking of Computational Devices

### 3.1.1 Purpose and Techniques

In Computer Science, "benchmarking" is the appraisal of the relative performance of computing devices by having a predetermined workload processed. The purpose of a benchmark [50] is to provide:

- A functional demonstration of the computing system
- Data for analytical study (*i.e.*, measurement)
- Data for the evaluation of performance (*i.e.*, prediction).

Evaluation of the system may involve the assessment of such specific performance metrics as the speed of integer and floating-point operations, ALU functions like matrix multiplication, memory address access rates, and disk operations.

The techniques for system evaluation are divided into classes by the type of workload they employ. The first of the two major types, "Application Benchmarking," uses actual programs on computer systems and records its performance. The desire for evaluation results that were "both easily understandable and based on several large real-life programs" was one of the motivating factors in designing program suites to do this such as SPEC [51]. The second, "Synthetic Benchmarking," utilizes software tools

specifically designed to test particular aspects of system performance. While this method does not give the workload-specific data that application program suites provide, (which is useful if the computer system's intended use is known) it is able to provide a more diverse range of possible uses by targeting multiple, specific operational aspects [52].

### 3.1.2 Strengths and Weaknesses of Benchmarking Techniques

There are a number of things that researchers have said both against and in favor of benchmarking.

Some of the drawbacks and challenges involve:

- Compatibility issues between the benchmarking software and the hardware of the system [52]
- The rapidly changing technology involved in computing systems, requiring evaluation software to keep pace [52]
- Potential inaccuracies in measuring computer system performance as compared to the actual examination of that system during its normal state of operation [53]
- Ensuring that the workload is properly reflective of the computer system's intended use [53]
- Bias in the interpretation and reporting of results [54]
- "Compiler effect" when compiling the code with software tools, which can significantly affect job speed [55].

On the other hand, benchmarking is considered to have significant advantages when properly conducted and documented, such as:

- Broad applicability to a variety of computer systems, allowing assessment by a common standard [53]
- A reasonable means for planning hardware purchases [55]
- A method for evaluating tradeoff decisions during the microprocessor design process [56].

### 3.1.3 Common Benchmarks

Common benchmarks and their functions, include:

- Whetstone – A set of software instructions geared toward numeric programs, which primarily tests floating-point arithmetic, as well as integer, trigonometric and other math library functions [57].
- Dhrystone – A program similar to Whetstone that focuses on integer operations, (it has no floating-point operations at all) and reports its results in the number of operation loops per second or, as another measure, the Dhrystone Million Operations Per Second (DMIPS) [57].
- Linpack – A program that consists primarily of linear algebra operations performed on a two-dimensional matrix of variables. The performance measure is reported in terms of millions of floating-point operations per second (MFLOPS) [57].
- SPEC – A set of suites that measure a large number of performance measures including CPU performance, graphics, client/server transactions, mail servers, power and performance, etc. [58][59].
- TPC – Software suites geared toward online business transactions that evaluate software/database interaction and data processing [58][60].

- BYTEmark – A CPU benchmark suite that measures CPU, cache and memory rate, as well as integer and floating-point performance [58].
- UnixBench – A Linux benchmark suite that primarily evaluates CPU and file I/O.
- EEMBC® – A benchmark suite specifically designed for embedded systems. [61] The version for testing Java implementations in mobile devices like cellphones and PDAs, focusing on the Connected Limited Device Configuration (CLDC) is called GrinderBench™. The results of the Java implementation benchmark are quantified as a measure known as the GrinderMark™, some examples of which [62] for current mobile device models include:
    - Motorola Razr V3, GrinderMark™ score: 131
    - Motorola V600, GrinderMark™ score: 140
    - Sharp GX-31, GrinderMark™ score: 265
    - Palm Treo 650, GrinderMark™ score: 1688.

## 3.2 Performance Measures for Evaluation

### 3.2.1 The Selection of Performance Measures

While the concerns and challenges regarding the use of benchmarks have been noted, they continue to be useful in the evaluation of complex and new hardware and software architectures as they become available. Cellphones, which represent some of the most recent examples of the miniaturization trend in computer system design, are also among the fastest growing market for computing devices. Providing a means for

evaluating relative performance of these many devices is a current [62] and ongoing source of interest for both consumers and industry.

Although a number of studies have been performed on mobile devices for evaluation purposes [65] a survey of the literature reveals that the work thus far done generally focuses on a few isolated factors. Rarely are the interacting components studied in concert, and almost never from an application developer's standpoint. More than a mere synthesis of previous work, this paper seeks to present a fresh look at the cellphones' computational abilities from the relatively unique viewpoint of an integrated system.

To accomplish this, the factors already discussed, along with the common benchmarks, must be evaluated in order to extract a set of reasonable performance measures for providing a means by which developers may better understand their devices' capabilities and limits in evaluating tradeoffs and software design decisions. The following set of measures is supplied in order to give a rounded view of the systems' performance in a variety of areas.

### 3.2.2 Power Consumption

Power consumption, as mentioned in Chapter 2, is a major consideration for software developers. Any ubiquitous or long-term application running on a device with resource constraints due to battery size and strength needs to take into consideration the length of time that the hardware is able to remain operational.

In addition, as the applications that may be run on a cellphone may access a wide variety of functions and hardware modules, such as the arithmetic-logic unit, the

secondary storage devices, the GPS chip and the wireless transmitter, the issue of power consumption is paired with the other factors described below to provide a set of values designed to give an estimate of how long the cellphone being examined is expected to run based upon the particular operations it is performing.

Developers will thereby have an idea of which operations and hardware functions constitute the largest drain upon the power reserves of their devices, and may seek ways to factor this into the design of their applications.

### 3.2.3 CPU Performance

Naturally, any analysis of a computing system is concerned with CPU performance. While it is true that efficient code and effective compilers may have a dramatic impact on the execution speed and throughput of running software, the ability of the system itself to process raw data is, in a way, the "bottom line" for how quickly coded instructions can be expected to execute.

Common measures of CPU performance [65] include IPS (instructions per second), FLOPS (floating-point operations per second) and the orders of magnitude associated therewith that are reported by prefixes on the existing algorithms. For example, IPS measured in millions is million instructions per second (MIPS) and the corresponding magnitude reported in floating-point operations is MFLOPS.

For the purposes of this framework, metrics directly related to MIPS and FLOPS were chosen due to their common use in industry. Both these measures have been the result of some criticism, primarily due to their perceived lack of reliability by some researchers [66]. Specifically MIPS, which is obtained by dividing executing code's

instruction count by its execution time in milliseconds, may be of various lengths in terms of that instruction count depending upon the hardware architecture for which it is compiled. For this reason, MIPS counts are not particularly useful across different instruction set architectures (ISAs).

MFLOPS, or their corresponding measure in billions, (GFLOPS) are somewhat more specific than MIPS, focusing on the execution time of a particular type of instruction – the floating-point operation. It faces a similar issue as MIPS, however, in that not all floating-point operations are implemented in the same way on all machines. For example, one machine may require only one instruction to perform a particular operation, while another may require two instructions to achieve the same result.

Despite the criticism, even those who point out the shortcomings of MIPS and FLOPS generally admit that these are among the most common means of reporting on CPU performance, and in cellphone architecture the variability found in larger, more complex machines may not be as much of a factor. This is one area in which the limited resources of cellphone devices may actually prove advantageous in the derivation and use of familiar, portable performance measures [67].

Within the software environment of the cellphones, (see Section 2.1.2.3) the Connected Limited Device Configuration (CLDC) specifies the basic libraries and virtual machine features that are actually implemented by the underlying hardware within the J2ME environment. While it supports a number of profiles tailored to more specific types of devices, the CLDC itself defines only the standardized, basic functions, while the devices supporting the Mobile Information Device Profile (MIDP) – which include cellphones and PDAs – do not have the diversity of architectures that may be present in

desktop and other, more complex, computing devices. Comparisons across CLDC and MIDP-compatible devices may thus be more consistent than comparisons across different PCs and other stationary (and more complex) systems.

### 3.2.4 Communication Elements

The key selling point of wireless computing devices is that they are not bound to specific locations. At the same time, as the complexity of applications increase, even those mobile computers with the capacity to perform advanced calculations are coming to rely upon communication with stationary devices. This is particularly true when dealing with distributed computing, or operations that involve interaction with a central database.

It goes without saying that the transmission of voice information is essential for the operation of cellphones, but even when viewed as a computing device independent of this primary feature, the wireless element of the architecture comes into play heavily for many of the applications that end users employ on their devices. If the processing work may as well be done on a stationary terminal, mobile computing loses much of its appeal. If, however, the results of calculations done on other machines can be transmitted to, and displayed upon, a mobile device, it alleviates some of the tradeoff of mobility for processing power, and the best of both worlds may be enjoyed to an extent.

The methods available to application developers, and the efficiency and reliability with which the devices can carry out these implemented procedures, goes a long way toward appraising the "mobile" element of mobile computing. The communication aspects of the devices in question determine how efficiently distributed calculations may be performed and analyzed.

### 3.2.5 GPS Hardware Performance

Interaction with the Global Positioning System is a lesser consideration than CPU performance, power consumption and communication in an overall sense, but it is becoming increasingly important to software developers. A look at the showcase of recently released application for mobile phones [48], some of which were mentioned in Chapter 2, reveals that most if not all of them involve navigation or other location-aware services on at least some level of functionality.

There are several considerations when dealing with the evaluation of GPS hardware embedded in computing systems. The level of performance depends to a large degree upon factors external to the device itself, including signal quality, geographic location, accuracy of the internal satellite clocks, clock discrepancies between satellites, and environmental obstructions [68]. When designing a framework that seems to evaluate the phone's performance in terms of location-aware functionality, the above factors must be held as constant as possible, as well as the hardware and software elements of the system.

Performance metrics of GPS performance are not common. Two of the variables associated with information derived from the positioning satellites, however, are the timeliness with which the data is obtained and processed by the receiving device, and the accuracy associated with the geographic location thereby derived.

#### 3.2.5.1 Fix Times

The first of these two measures is the "Fix Time," and like the other metrics associated with GPS performance it is strongly influenced by the variables listed above.

In order to provide a benchmark that may be evenly applied across hardware devices, a number of additional factors must also be taken into consideration.

There are three distinct kinds of fixes that may be obtained by the receiving GPS device [13]. A "Cold Fix" is that set of position data that is obtained at intervals of an hour or longer. These are also the first fixes retrieved upon initial activation of the GPS chip. In order to obtain a cold fix, therefore, an hour must be allowed to elapse between fix requests, or the phone must be powered down and restarted. This kind of data is utilized by applications that require location data only once in an extended period of time.

A "Warm Fix" is obtained between ten seconds and an hour of a previous fix. This kind of GPS data is that which is utilized by applications requiring regular but non-continuous location updates. A "Hot Fix" is the result of continuous GPS chip activation, and is obtained within ten seconds of a previous fix. This data is required by applications that require rapid real-time location data and instant interaction with the receiving device.

For the purposes of this framework, and with the many external factors that may influence its results, cold start fixes are unlikely to yield a proper performance measure. In the hour it takes to await this fix, the environmental conditions may have changed to the degree that the time to receive and calculate data may be altered. This action, as well as restarting the hardware completely, also requires a new detection routine to be run by the device in order to determine the available satellites for signal reception, and this time is also variable.

Hot start fixes tend to come fairly constantly once the GPS chip is in an activated state, and would thus yield little by way of variability for comparison. In addition, applications that require constant location data from receiving devices are of limited

applicability in cellphones, since the accuracy associated with the fixes from these devices makes the ability to utilize such fine-grained information suspect. The large increase in power consumption associated with continuous hardware activity (as revealed in Chapter 5) is another factor that would limit the usefulness of hot fixes in practical applications.

The rate at which warm start fixes may be obtained by the mobile device was chosen as a metric. These fixes, which provide periodic updates that are separate enough to be useful, yet frequent enough to provide enough data to reasonably track a mobile device's travel route, are also useful for study when considering power consumption. The cold fixes are too infrequent to show a proper degree of decrease in battery life over time. Hot fixes, as mentioned above, would reveal the result of continuous chip activation, and would not reflect a realistic rate of declining battery life in any but the most specialized of applications.

#### 3.2.5.2 Estimated Accuracy Uncertainty

Accuracy in GPS receivers is not provided as a degree of confidence in the location reported. Instead, a metric called the "Estimated Accuracy Uncertainty" is associated with each location fix. This value, illustrated in Figure 3.1, is a radius in meters that describes a circular area around the reported latitude and longitude of a geographic fix. In the Location API available to J2ME enabled cellphones that are GPS enabled, the value indicates that the location reported is located within an area of this radius with a 68% degree of confidence [69].

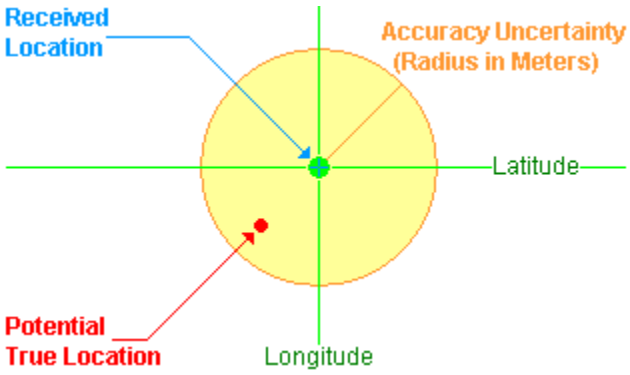


Figure 3.1 – Estimated Accuracy Uncertainty

While this is not an extremely high confidence interval for the uncertainty estimate obtained, it provides at least some measure of device-reported performance, and is thus a reasonable metric for the purpose of comparative evaluation with other devices that use the same API.

### 3.3 Unique Factors in the Evaluation of Mobile Phones

With the metrics for this proposed framework identified, these being CPU performance, communication reliability and efficiency, and location-based functionality, as well as the power consumption involved with these processes, we now know what to measure, and why. "How" is the next question, and while the specifics of this are largely dealt with in the following chapter, some factors bear mentioning here on the heels of the discussing justifying this selection of metrics.

Since the architecture in a cellphone, on its most basic level, is identical to that of a stationary computing device, the measurement of CPU performance requires no special terminology or methods. Units such as "Instructions Per Second," and "Floating Point Operations Per Second" are easily understandable for those in the development or

manufacturing fields, and while the tools may be more refined to run on the relatively limited resources available, the underlying methodology is the same: provide a known workload and evaluate the time required to process it.

Communication is certainly a noteworthy aspect of any computing device, mobile or otherwise; however, since the transmission and reception of data by mobile devices are almost entirely reliant upon the wireless networks to which they are connected the focus of the performance metric is different. For the purposes of this framework, the methods proposed are based upon time-of-transmission, and not upon the specifics of the communication method employed. In other words, the same method for evaluating communication functions may be employed for HTTP, JAX-RPC and other wireless protocols, as long as the method is identified along with the metric. The one-way UDP transmissions require additional consideration for reasons discussed in Chapter 4.

While power consumption is also studied for stationary devices, this is not as critical for immobile computing systems. The monetary costs of operation, and the heat generated by use of power, (which is higher with inefficient hardware) are issues that have been and are being studied [69], but mobile devices are entirely limited in their ability to function by the life of a portable battery, and how costly the functions they are expected to perform prove to be in terms of energy use. For this reason, the framework proposed not only measures power consumption as an isolated factor, but also as an related aspect of each of the other three types of measures being reported.

Location-aware aspects of computing are almost completely restricted to mobile devices. Obviously, stationary devices do not require constant updates regarding their geographic location, and related applications are not commonly designed for such

systems. All of the GPS-related factors that relate to performance measures described in Section 3.2.5, therefore, apply almost exclusively to mobile computing.

What we find, then, is that as we go from aspects of mobile computing that are most like that done on stationary systems to those that are unique to cellphones, the methods for evaluating performance become more specialized. This is what one would expect, but it bears mentioning since the availability of tools and comprehensive frameworks like that proposed in this dissertation are neither common nor standardized. The more work done in this area, the more knowledge will become available to third-party application developers, and ultimately the more useful and effective the applications that will become available for consumer use.

# Chapter 4

# Framework Design

## 4.1 Design Considerations

The framework proposed by this dissertation is a structured method of system evaluation dealing with all the basic functions of the device that are critical to its abilities as a computational system. In designing this framework, a number of key factors merited consideration in addition to those mentioned at the end of Chapter 3. The choice of a software environment in which to develop the evaluation tools was a significant one, as compatibility with a wide range of devices is required, along with ease of use. Section 4.2 discusses this element of the framework.

One of the first and most important challenges was the selection of metrics to record and analyze, since these constitute the very core of the proposed framework. With so little consistently quantified in cellphone technology, a wide range of possibilities existed. In the interest of feasibility and usefulness, a very narrow subset of all the possible measures (*i.e.*, specific measures for CPU performance, communication methods, etc.) needed to be selected while maintaining a fair representation of the devices' capabilities. Section 4.3, Section 4.4, Section 4.5 and Section 4.6 describe the parameters chosen. Section 4.7 provides the resulting table of values.

Section 5.1 within the next chapter discusses the selection of a phone model to which this framework was applied in order to provide an example of its fundamental value. The phone selected needed to meet a number of general criteria. It had to:

- Be of fairly recent design
- Provide a Java software environment accessible to third-party developers
- Support GPS functionality that may be employed in user applications
- Permit developer access to wireless transmission capabilities
- Contain persistent memory storage for retaining data about battery depletion rates after shutdown (see APPENDIX A).

### 4.2 The Application Development Environment

A proper discussion of performance metrics, and how they are calculated, necessarily involves an examination of both the software tools employed to obtain those metrics and the degree of similarity of the simulated workload to what the benchmarked devices are expected to do in practice. In the proposed framework, several of the tools used to perform benchmarking operations were created in a common application development environment, and this ensures, to a large degree, an applicability of the benchmark results to real-world operations. The same software compilers and program libraries are involved, thus any optimization procedures that may be automatically applied to program code would be uniformly applied to the benchmarking applications and those produced for consumer use.

The NetBeans Integrated Development Environment (IDE) allows for relatively rapid and easy development of mobile applications, including both the client-side and

server-side elements of those systems that involve wireless communication. One of the most obvious benefits over older software tools is the ability of NetBeans to allow users to develop, deploy and test J2ME applications all in the same environment, emulating the remote computing device as well as implementing the communication and server-side aspects of the system.

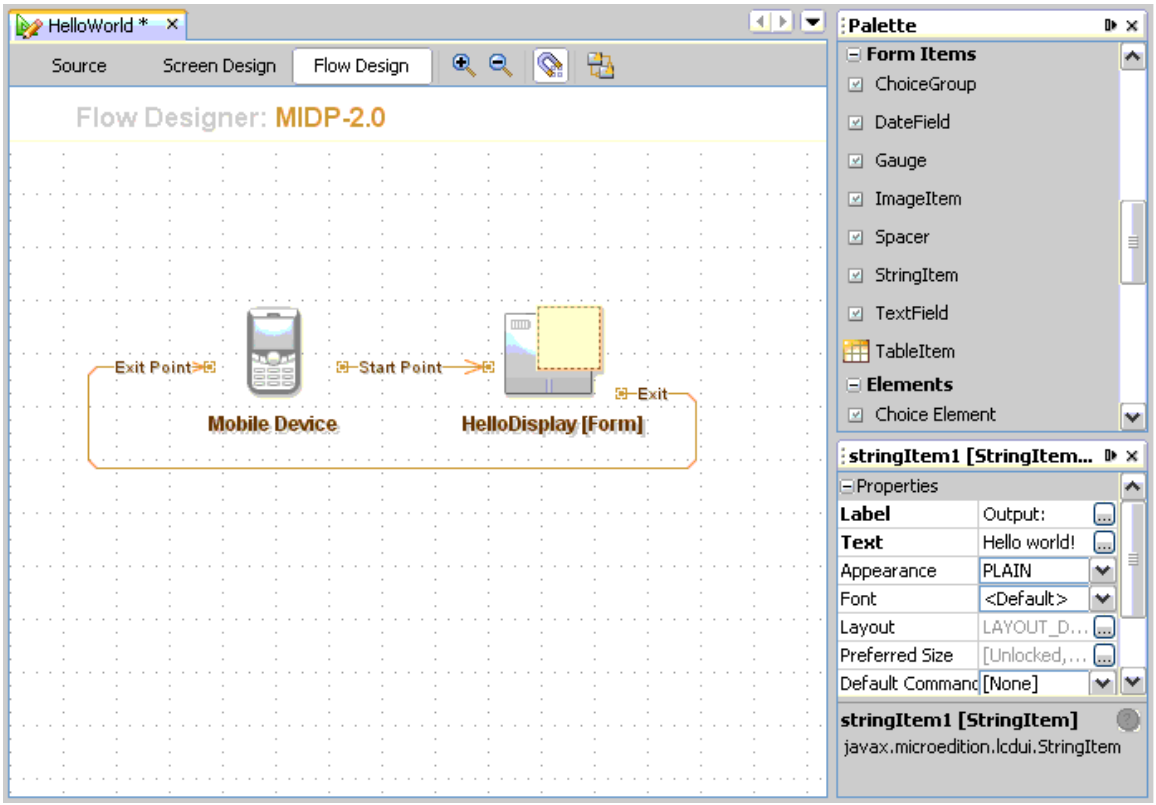


Figure 4.1 – The NetBeans IDE Interface

Designing the user interface for new applications is rendered intuitive by the Flow Design and Screen Design features, (Figure 4.1) allowing developers to concentrate almost exclusively on underlying functionality and whatsoever communication functions the software requires.

Because NetBeans is easy to use, able to develop applications rapidly, and is available free of charge, (thus accessible to every developer) it was chosen as the environment in which to develop the tools necessary for this evaluation framework. Updates and plugins available on developer sites allow applications to be produced for a wide variety of mobile phones across both models and manufacturers, and this facilitates the production of tools that are highly platform-independent, with the possibility of easy adaptation to new models as they become available.

### 4.3 Evaluating Power Consumption

A "baseline" measure of energy consumption may be obtained by simply charging the mobile device to its full capacity and leaving it to run until the battery is exhausted. The starting point and ending points of the test run are recorded by observation, and the time difference is recorded as the battery life. This simple method, by itself, is of little value, since even the most trivial of computing operations (including the normal communication functions that a phone is expected to perform) hasten the rate of decline. It does, however, provide a starting point for the evaluation of power consumption rates when known workloads are being processed.

No specialized software needs to be developed for obtaining this value; however, as will be discussed in detail in the sections that follow, pairing power consumption rates with the other performance measures allows researchers to consider the device on a system-wide level. The power consumption aspect of a mobile computing device, which may often be ignored entirely when being run on a stationary computing system, is highlighted in this framework.

A cautionary note must be added when discussing the comparison of device power consumption rates that are measured by observing effective battery life. Not only does battery strength decline with use between recharging periods, but overall maximum battery life is also subject to a number of external factors that accumulate with use [71]. In other words, devices that are identical in every other aspect may operate for different lengths of time, even on a "fully" charged battery, if one of the devices has been in service longer than the other. It is important to this aspect of the framework, therefore, that the devices being tested are equipped with batteries that are as new as possible in order to provide a like basis for comparison.

### 4.4 Evaluating CPU Performance

Although – perhaps because – cellphones provide a unique computing and development experience, it is important to present such vital performance measures as CPU performance in terms that are readily understandable to potential users and application developers. Two better-known measures selected for this framework (the DMIPS and MWIPS quantifications) may be measured using common software tools designed for that purpose (see APPENDIX B).

The Dhrystone benchmark is employed to arrive at a value called DMIPS [57], which is essentially a measure of how many integer operations the system may perform in a given period of time. The benchmark program itself provides a figure representing the number of code iterations per unit of time, (*i.e.*, "Dhrystones per second") and this value is then divided by the number 1,757. This divisor is an index constant obtained by running the Dhrystone operations on the VAX 11/780, a machine that was measured to

run at 1 million instructions per second (1 MIPS) [72]. Since the VAX 11/780 performed 1757 Dhrystones per second at 1MIPS, the DMIPS value is obtained by dividing the Dhrystone result of a system being evaluated by the VAX result. This value continues to be used in industry to describe CPU performance.

Whetstone is used to derive the floating-point rating; specifically, the measure of floating-point operations provided by Whetstone is called WIPS (Whetstone Instructions Per Second), and may be recorded in the millions as MWIPS. Commonly available JAVA versions of these software applications [73][74] were adapted to run in J2ME for the purposes of this framework, with an additional file [75] imported into the Whetstone adaptation to facilitate certain mathematical functions (*i.e.*, exp, log, atan) not available in the standard MIDP implementation.

In addition, the EEMBC benchmark suite discussed in Section 3.1.3 may be effectively utilized due to its specialized function for the evaluation of embedded, Java-enabled systems. While not yet used as a common standard, the developers of this suite have already conducted publicly reported evaluations of a number of popular mobile device models, and the data may thus be useful for comparison against the cellphones tested by this dissertation's framework.

Energy consumption is measured by placing the Dhrystone and Whetstone operations into infinite execution loops, and then recording the time taken for the phones running these applications to entirely deplete their battery power when starting from fully charged states.

### 4.5 Evaluating Data Transfer Capabilities

Of the various methods of wireless data transfer described in Section 2.2, two of the most fundamental are HTTP and UDP. HTTP is a common request/response protocol that is commonly employed in the wireless client/servlet system. As it can be employed on top of any reliable protocol, such as TCP, it may be implemented across a wide variety of software platforms. In the software environment used to develop the tools for this framework, an HTTP based method known as "Representational State Transfer" (REST) is the architectural style upon which the web services used for transferring some of the data between the client device and the remote client is built.

The latency of message transfer using this method, which may be described as RESTful HTTP, (*i.e.*, an HTTP-standard-based method compliant with the REST architectural style) depends upon the design of the communication protocol as well as the format of the data being transmitted [76]. One of the benefits of using RESTful HTTP is its ease of implementation in the NetBeans environment. Upon the development of a mobile application's server-side software, the client application may be specifically described within the environment as a "Mobile client to web application," and the communication protocol is automatically prepared for use. Additionally, RESTful communication methods are purported to be more reliable, more scalable and more secure than other existing web service communication techniques [77], contributing to the probability of its widespread use in future mobile applications, and the continued relevance of its inclusion in an evaluation model such as that which is described in this dissertation.

UDP, which does not require the reception of acknowledgements involved in HTTP style communication, not only finds use in different kinds of applications (or functions thereof) but also provides a completely independent performance measure due to its fundamentally dissimilar protocol. In the server-side software of a UDP system, a UDP Listener, which is a software agent designed to detect incoming transmissions, is activated. Upon receiving a message via this method the listening agent makes the data thus obtained available to the other server functions.

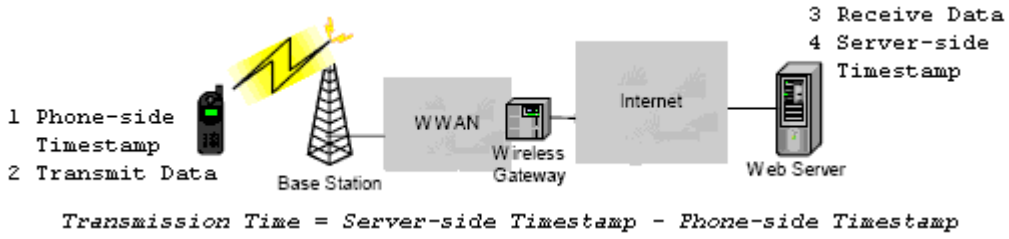


Figure 4.2 – Wireless Transmission Time

The basic method for evaluating wireless transmission time involves the use of the time stamp that may be associated with each package of broadcast data. Since recent cellphones receive their local time value from network data [78], (as opposed to being set manually by the user) this value was assumed to be fairly accurate. A simple application for testing this value is illustrated in Figure 4.2, which involves (1) having the mobile device ascertain the local time (2), transmitting a burst of data wirelessly via the method being evaluated, and (3, 4) recording the timestamp of data reception on the server.

In practice, however, where transmission times may be less than a second, as was consistently observed in UDP transfer testing, even slight differences in synchronicity

between the client and the server clocks can lead to significant inaccuracies in measuring the transfer time. Several of the preliminary tests done indicated that the server received the UDP data before it was transmitted – indicating an obvious discrepancy of a large enough magnitude to completely invalidate any measurements.

Since no methods for synchronizing these clocks are readily available to third-party developers, for whom this framework is intended, an alternative method (illustrated in Figure 4.3) was implemented. Instead of focusing on total transmission time for the UDP, the scope of the metric is limited to the phone-side processes only. The "UDP transmission time" of the framework then becomes the time it takes for the transmitting device to open the wireless data channel, send a data packet of pre-determined size, and close the data channel, freeing up the communication resources and allowing operation to proceed for any client functions awaiting the completion of the transmission sequence.

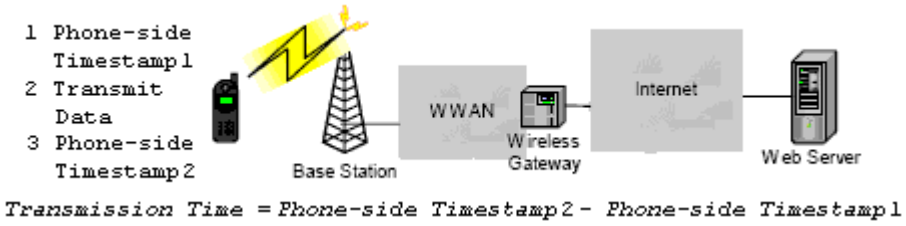


Figure 4.3 – Evaluating Phone-side Transmission Time

For send-receive methods such as RESTful HTTP, synchronizing the clocks of the phone and the server are not important, since the transmission cycle ends when the phone obtains a response. Timestamping the phone before sending the data and again

after receiving an expected acknowledgement (see Figure 4.4) provides an accurate method of measuring the total transmission time.

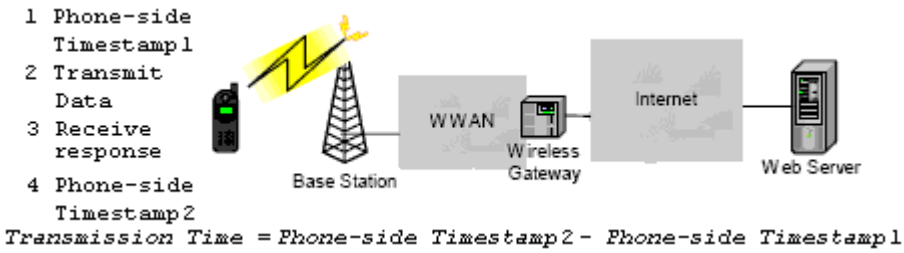


Figure 4.4 – Evaluating Send-Receive Transmission Time

The data transmitted for the purposes of this framework in both the UDP and RESTful HTTP methods takes the form of the following string:

*<9999;999;99.999999999999999;0;99.999;9999999999999;99.9;999.99;999.99;999.99;9;9;9999999999999;false;9;9;9;9999>*

This combination of characters, which simulates data transmitted from a GPS-enabled cellphone to a remote server that is tracking the device's movements, was chosen for testing both methods used in this evaluation, for a number of reasons:

- GPS data realistically models the workload of actual projects [79] that utilize frequent wireless transmissions.
- It is of sufficient length to represent the exchange of a significant amount of information.
- It is a small enough data block to enable transmission in a single packet, which ensures that only a single instance of transmission will be involved in the transfer.

The software developed for testing UDP and HTTP data is identical except for the procedures required to establish the respective communication protocol and transfer the data. APPENDIX C contains a description of the code used for this purpose.

Power consumption of the HTTP and UDP methods is measured by means of transmitting the string described above repeatedly until the battery is completely depleted and the phone shuts down. Upon each data transmission event in the application, if the battery level of the device – an integer value that may be obtained within the software environment – has declined, (*e.g.*, from 4→3 or 3→2) the timestamp of that transmission is recorded along with the associated battery level (see APPENDIX A). This measure is taken with data transmitted over the two protocols at 15 second, 30 second and 60 second intervals in three trial runs, enforced by means of a Sleep method in the transmitting program thread. Although a battery degradation graph (*e.g.*, Figure 4.5) is obtained from each of these intervals, which shows the duration of each battery level, only the total battery life duration is reported as the metric for comparison.

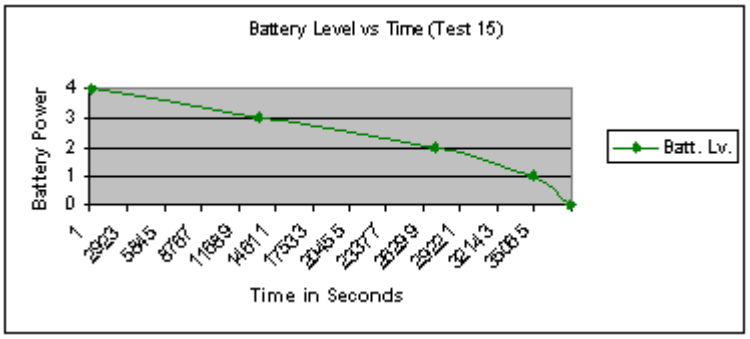


Figure 4.5 – Example of a Battery Life Degradation Graph

For practical purposes, a cellphone will continue to function until its charge is entirely depleted; therefore from an application restriction standpoint only the final timestamp is of any great significance.

### 4.6 Evaluating Location-based (GPS) Data Manipulation

While recent research has revealed that the choice of software interface interacting with the GPS hardware of a cellphone has a dramatic influence upon its location-aware capabilities [80], it is also the case that some cellphones are more sensitive to the satellite signals of the GPS system than others. This research also revealed that the time-out value a user may provide in the software for requesting a valid fix has relatively little influence on the timeliness of data reception, therefore a simple application was developed to allow time-stamping the initial GPS fix request and then the time at which the data was obtained. The following code within the application (see APPENDIX D for a more detailed description of the software tool) provides the requests for GPS fixes at *X*-second intervals, where *X* is between 10 seconds and an hour to ensure "warm start" fixes:

```
FixTimeA= System.currentTimeMillis();
location = lp.getLocation(X);
TimeDiff = System.currentTimeMillis() - FixTimeA;
[...]
```

The following code included in the application also provides another performance measure, the estimated accuracy uncertainty associated with each fix:

```
Accuracy = location_data.getHorizontal_Accuracy();
```

in which *location_data* is an object of type "JSR179LocationData" (the Location API standard) obtained from the location returned by the above code.

Two sets of data are involved in the reporting of these metrics. The first is the average fix-time interval and estimated accuracy uncertainty in an open area with the device having a clear view of the sky, and thus easy access to the orbiting GPS satellites. The second set consists of the fix time interval and accuracy values taken indoors, with solid walls obstructing satellite access. This measure is taken within concrete walls, five feet (60cm) away from a glass window. The final values reported are an average of the warm-start fixes and estimated accuracy uncertainty of a hundred trials conducted under the two conditions described.

Power consumption associated with obtaining GPS coordinates is evaluated using a different application, a version of the second application described in Section 4.5 with its settings adjusted. Specifically, this application, while transmitting dummy values when evaluating the power consumption associated with the HTTP and UDP methods, is also capable of obtaining actual GPS coordinates and transmitting them instead. As before, the application records the times at which the battery level of the transmitting device decreases in the cellphone's persistent on-board memory; this allows the measure of battery power degradation to be obtained. Naturally, since two distinct procedures are taking place simultaneously (both obtaining GPS data and transmitting information wirelessly) further analysis must be performed to obtain the rate at which just obtaining GPS coordinates contributes to the overall power consumption.

Since static data is transmitted in the RESTful HTTP and UDP evaluation of Section 4.5, recording the battery power at the time of a decline in level reflects only the

power consumed by the transmission process (as it accelerates the natural battery life decline).

The application can be modified to operate in three different modes, thus providing other measures for evaluation:

- Mode 1: The application transmits dummy GPS coordinates over HTTP or UDP at specified intervals. This mode is used as described in Section 4.5 to obtain the power consumption of wireless transmission only.
- Mode 2: The application receives GPS data at specified intervals but does not transmit information to the server. This mode is used to obtain the power consumption for just obtaining position coordinates.
- Mode 3: The application receives GPS coordinates at specified intervals and transmits them to the remote server over HTTP or UDP. This provides the combined metrics GPSHTTP*X* Battery and GPSUDP*Y* Battery, where *X* and *Y* are the intervals (15s, 30s and 60s) used for the HTTP and UDP protocols respectively.

This last set of metrics best simulates an actual location-aware application's workload. The full set of metrics is described in the following section, and Figure 4.6 provides a conceptual illustration of the framework's areas of focus. Each large box of tie diagram contains the primary metrics of an aspect of the mobile phone's capabilities, and each small box contains the combined metrics derived from the larger boxes with which it is associated (*e.g.*, "GPS Power Consumption" contains the measures derived from the "Power Consumption" and "GPS Functionality" primary metrics).

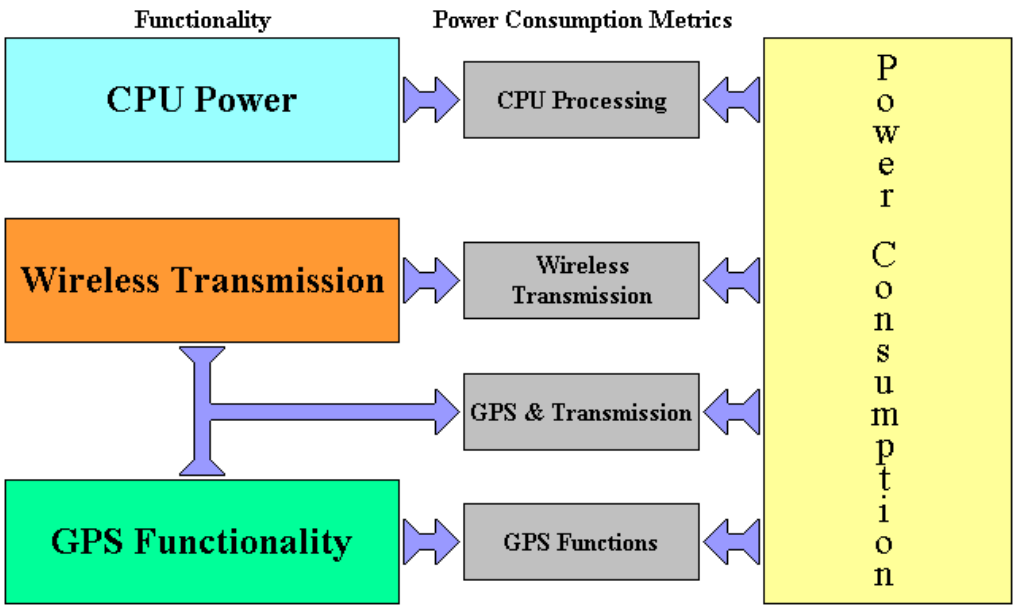


Figure 4.6 – A Conceptual View of the Framework

## 4.7 The Table of Metrics

A table of the metrics described by this framework is given in Table 4.1.

Table 4.1 – The Table of Performance Measures

| Phone Model | Processor | Firmware | Network | Programming Language | Baseline Battery | DMIPS | DMIPS Battery |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| MWIPS | MWIPS Battery | Grinder Mark | HTTP Time | HTTP15 Battery | HTTP30 Battery | HTTP60 Battery | UDP Time |
| | | | | | | | |
| UDP15 Battery | UDP30 Battery | UDP60 Battery | FixTime Clear | FixTime Obscured | EAU Clear | EAU Obscured | GPS15 Battery |
| | | | | | | | |
| GPS30 Battery | GPS60 Battery | GPSHTTP 15 Battery | GPSHTTP 30 Battery | GPSHTTP 60 Battery | GPSUDP 15 Battery | GPSUDP 30 Battery | GPSUDP 60 Battery |
| | | | | | | | |

The Phone Model is the manufacturer and model number of the specific phone being evaluated. The Processor hardware is its main computing component, while

Firmware is the resident controlling program of the processor, and acts as an interface between the phone's hardware and software. The Network value indicates the company supporting the wireless communication functions of the phone.

Baseline Battery is the time (in seconds) that it takes for the battery power to be depleted from a full charge to shutdown with no third-party applications active. DMIPS is the DMIPS rating and MWIPS is the MWIPS rating of the device, while DMIPS Battery and MWIPS Battery are the battery life durations (in seconds) with the DMIPS and MWIPS evaluation programs (respectively) running in an infinite loop. GrinderMark is the GrinderMark™ metric returned by the GrinderBench™ benchmark of the EEMBC® suite.

HTTP Time is the time it takes to transmit data using the RESTful HTTP method, averaged over a set of hundred trials. HTTP15 Battery, HTTP30 Battery and HTTP60 Battery are the battery life durations of the device when HTTP data is transmitted with 15 second, 30 second and 60 second intervals respectively. UDP Time is the time it takes to transmit data using the UDP method from the client device's perspective, likewise averaged over a set of hundred trials. UDP15 Battery, UDP30 Battery and UDP60 Battery are the battery life durations of the device when UDP data is transmitted with 15 second, 30 second and 60 second intervals respectively.

FixTime Clear is the interval between requesting a GPS fix and obtaining it when the device is relatively unobstructed by environmental factors, and the value reported is the average duration of a hundred fixes. FixTime Obscured is a similar measure, but taken indoors as described in Section 4.6. EAU Clear and EAU Obscured are the average

Estimated Accuracy Uncertainty values associated with the “Clear” and “Obscured” fixes obtained. These are also average values obtained over a hundred trials.

GPS15 Battery, GPS30 Battery and GPS60 Battery are the battery life durations of the device when GPS data is obtained using 15 second, 30 second and 60 second request intervals respectively. Finally, GPSHTTP15 Battery, GPSHTTP30 Battery and GPSHTTP60 Battery, GPSUDP15 Battery, GPSUDP30 Battery and GPSUDP60 Battery are the battery life durations of the device when GPS data is both obtained and transmitted via the HTTP and UDP methods using 15 second, 30 second and 60 second intervals respectively. In order to ensure the best possible satellite access, the phones were placed outdoors for these last six tests. Despite the presence of buildings, even an urban environment provides greater GPS accuracy (see Chapter 5) if the receiving devices are not enclosed by solid walls.

# Chapter 5

# Framework Evaluation of a Mobile Phone

## 5.1 Selection of a Mobile Phone

Several models of mobile phones met the criteria for consideration outlined in Section 4.1, and were therefore considered for producing the case-study evaluation results of this framework. One in particular, the Motorola iDEN i580 (Figure 5.1 [82]) has been in use in a number of relevant research projects for several months, and is of a fairly recent design, having been released in April of 2006 [81]. The i580 model has the additional benefit of supporting two separate methods of obtaining GPS fixes from satellite data, these being the Motorola-specific Position API and the platform-independent Location API, which lends itself to API comparisons.

Figure 5.1 – The Motorola iDEN i580

A drawback of the iDEN series, however, is that it utilizes Time Division Multiple Access (TDMA) technology for its communication operations, a system that has been declining in use according to the trends in recent sales and its limited use in current and projected third generation mobile devices [83][84]. With newer phone models likely to continue with the use of the alternative Code Division Multiple Access (CMDA) technology [83], it is reasonable that evaluation tests of a framework intended for application to upcoming devices use a mobile phone incorporating this system.


Figure 5.2 – The Sanyo SCP-7050

The SCP-7050 by Sanyo (Figure 5.2 [85][86]) is one such model. It has all the relevant features of the iDEN model, including GPS and Java functionality, and it has the added benefit of utilizing the CDMA network [87]. It was therefore selected for the case study.

### 5.2 Power Consumption Evaluation Results

The baseline battery consumption for the Sanyo SCP-7050 was 1537200; with no applications running, the phone will go from a fully charged state to inactive in approximately 1,537,200 seconds (17 days, 19 hours). As this value was obtained by observation, a precise value (*i.e.*, to the very second of deactivation) was impractical; however, the margin of error provided by observing the phone at set intervals (3.5 hours) amounts to only 0.82% of the observed time. The battery life durations associated with the calculation of the other performance measures are provided in the appropriate sections below.

### 5.3 CPU Performance Evaluation Results

DMIPS were measured by means of a J2ME adaptation of the Dhrystone benchmark [73]. During the execution of the benchmark code, the cellphone was connected to its charger in order to maintain a consistent battery level during operation. The code was executed ten times, resulting in the following Dhrystone-per-second scores: 398879, 397594, 397750, 397717, 397632, 397852, 398413, 395989, 396011 and 396044. The standard deviation of these scores is 1026.86, and the average, 397388, is divided by 1,757 [72] in order to obtain the DMIPS value (226) recorded in Table 5.1. The battery life measured on the cellphone with the MIPS program running continuously in the background (DMIPS Battery) was 376,456 seconds, with the decline illustrated in Figure 5.3.

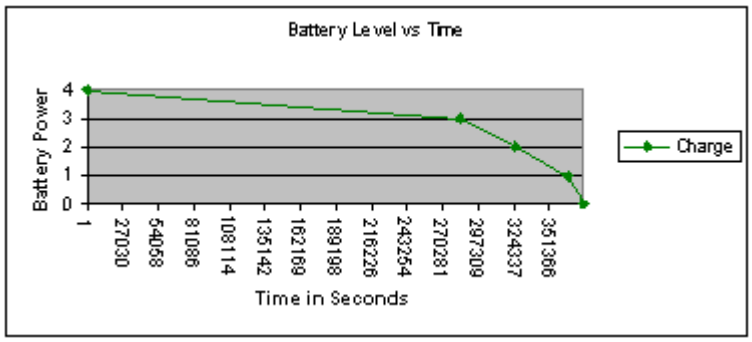


Figure 5.3 – DMIPS Power Consumption

A J2ME adaptation of the Whetstone benchmark [74] was used to measure the CPU speed when handling floating-point operations. As before, the cellphone was connected to its charger in order to maintain a consistent battery level during operation. The code was executed ten times, resulting in the following MWIPS scores: 158, 158, 157, 158, 158, 158, 158, 158, 158 and 157. The standard deviation of these scores is 0.4216, and the average, 158, is recorded in the MWIPS field of Table 5.1. The battery life measured on the cellphone with the MWIPS program running continuously in the background (MWIPS Battery) was 442,707 seconds, with the decline illustrated in Figure 5.4.

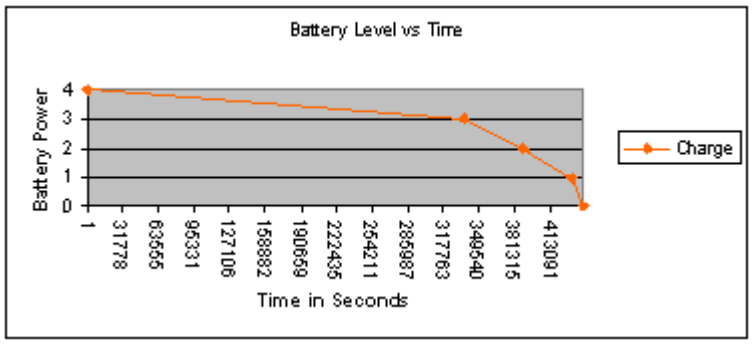


Figure 5.4 – MWIPS Power Consumption

The GrinderBench™ program from EEMBC was run ten times, resulting in the following ten scores: 3269, 3339, 3344, 3348, 3347, 3347, 3514, 3348, 3348, and 3346. The standard deviation was 60.93, and the average score of the ten trials was 3355. This average is recorded in Table 5.1 as the GrinderMark value.

## 5.4 Data Transfer Capabilities Evaluation Results

### 5.4.1 HTTP Time and Power Consumption

In a run of a hundred trials, with ten-second intervals between transmissions, the average time taken by the Sanyo cellphone to transmit the test string to a corresponding server using the RESTful HTTP method was measured. This average, 4261.31 ms with a standard deviation of 1823.79, is recorded in Table 5.1 as the HTTP Time score.

The duration of battery life while transmitting data via HTTP for three intervals, 15 seconds, 30 seconds and 60 seconds, is shown in Figure 5.5. The values were measured as 34405 seconds (over 1759 transmissions), 59783 seconds (over 1659 transmissions) and 78771 (over 1183 transmissions) seconds respectively for the intervals listed.

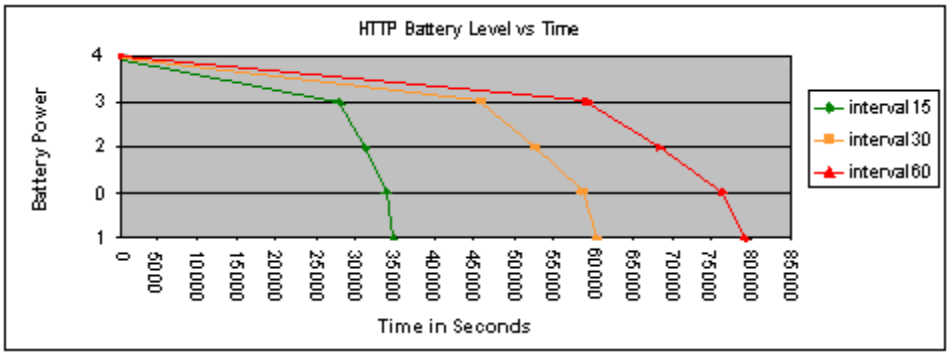


Figure 5.5 – Battery Life Decline vs. Time for Set Intervals (HTTP Data)

### 5.4.2 UDP Time and Power Consumption

In a run of a hundred trials, with ten-second intervals between transmissions, the average time taken by the Sanyo cellphone to transmit the test string to a corresponding server using the UDP method was measured. This average, 1523.7 microseconds with a standard deviation of 1584.75, is recorded in Table 5.1 as the UDP Time score.

The duration of battery life while transmitting data via UDP for three intervals, 15 seconds, 30 seconds and 60 seconds, is shown in Figure 5.6. These values were measured as 32507 seconds, 61270 seconds and 108089 seconds respectively for the intervals listed. Figure 5.6 shows the relative battery lives of these three trials.

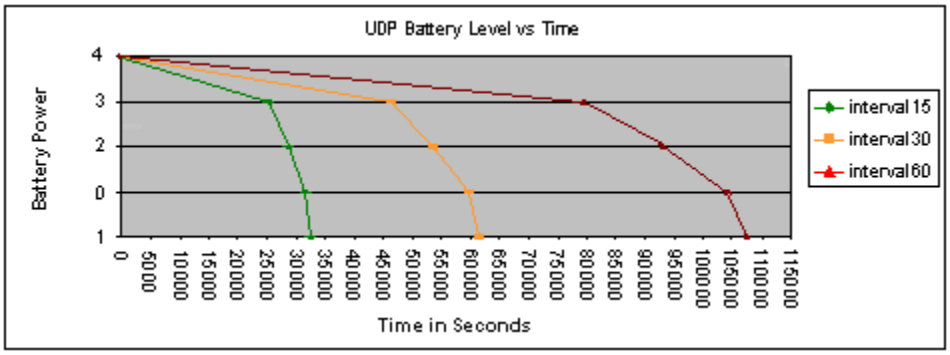


Figure 5.6 – Battery Life Decline vs. Time for Set Intervals (UDP Data)

## 5.5 Location-based (GPS) Data Reception Evaluation Results

### 5.5.1 GPS Reception Functionality

The time taken to obtain a GPS satellite fix was measured by means of the evaluation application described in Section 4.6. The results taken in an outdoor, urban area yielded a hundred fix times that display a standard deviation of 0.94. The average of

these scores, 1.77 seconds, was recorded in Table 5.1 under the Fixtime Clear field. The estimated horizontal accuracy uncertainty associated with each of these fixes, in meters, was recorded as exhibiting a standard deviation of 0.06. The average of the hundred values, 24.94 meters, was recorded in the EAU Clear field.

The results taken indoors, five feet away from a glass door, produced a hundred fix times with a standard deviation of 3.95 and an average of 8.497 seconds, recorded as Fixtime Obscured in Table 5.1. The estimated horizontal accuracy uncertainty associated with each of these fixes, in meters, was recorded as exhibiting a standard deviation of 35.29. The average of the hundred values, 81.63 meters, was recorded in the EAU Obscured field.

### 5.5.2 GPS Functions and Power Consumption

The first set of tests of both battery life and GPS functionality involves the mobile device merely receiving location information from positioning satellites. The phone does not transmit any information wirelessly. The GPS15 Battery value was measured at 52993 seconds. The GPS30 Battery value was measured at 47756 seconds. The GPS60 Battery value was measured at 44388 seconds. Figure 5.7 shows a graph of the battery power as it declines to 0 for each of the intervals examined.

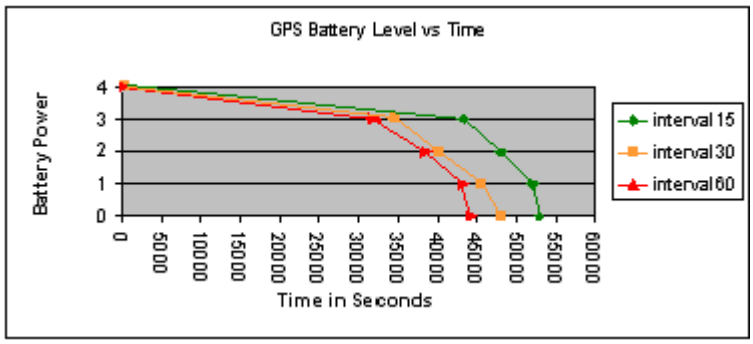


Figure 5.7 – Battery Life Decline vs. Time for Set Intervals (GPS Data)

The second set of tests considering battery life and GPS functionality involves allowing the mobile phone to both receive location data and transmit it to a remote server using the RESTful HTTP method with intervals of 15 seconds, 30 seconds and then 60 seconds. The GPSHTTP15 Battery value was measured at 43448 seconds. The GPSHTTP30 Battery value was measured at 64518 seconds. The GPSHTTP60 Battery value was measured at 80152 seconds. Figure 5.8 shows a graph of the battery power as it declines to 0 for each of the intervals of this test.

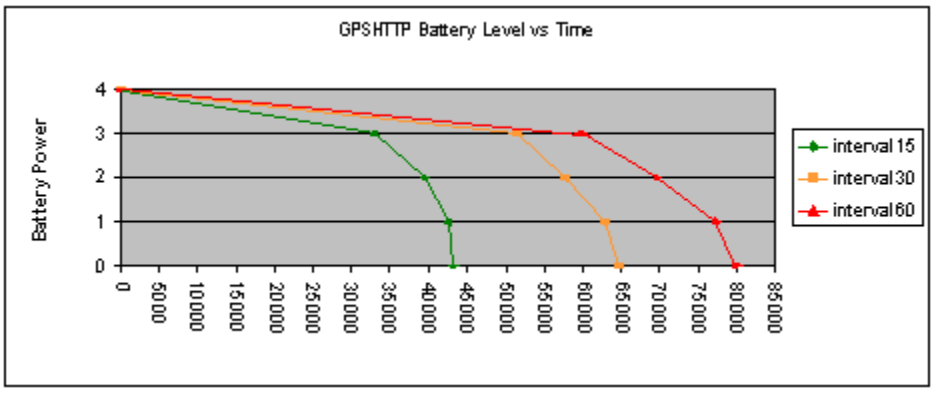


Figure 5.8 – Battery Life Decline vs. Time for Set Intervals (GPSHTTP Data)

The third set of tests considering battery life and GPS functionality also involves allowing the mobile phone to both receive location data and transmit it to a remote server. This time the UDP method is used at intervals of 15, 30 and 60 seconds. The GPSUDP15 Battery value was measured at 25501 seconds. The GPSUDP30 Battery value was measured at 37654 seconds. The GPSUDP60 Battery value was measured at 55193 seconds. Figure 5.9 shows a graph of the battery power declining to 0 for the intervals examined in this last set.

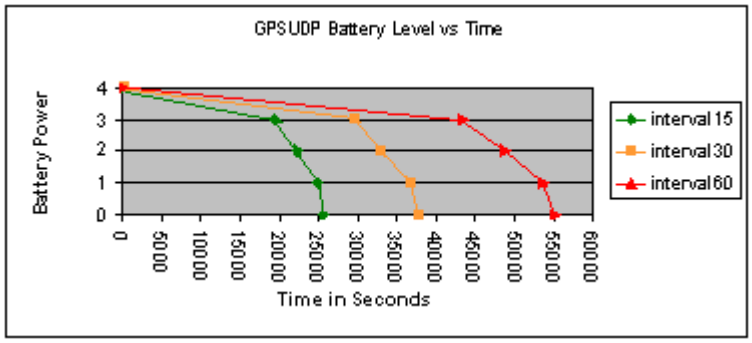


Figure 5.9 – Battery Life Decline vs. Time for Set Intervals (GPSUDP Data)

## 5.6 Framework Evaluation Result Summary

A summary of the application of the framework described by this dissertation to the Sanyo SCP-7050 cellphone (Section 5.2 – Section 5.5) is given in Table 5.1 below.

Table 5.1 – Table of Evaluation Results

| Phone Model | Processor | Firmware | Network | Programming Language | Baseline Battery | DMIPS | DMIPS Battery |
|---|---|---|---|---|---|---|---|
| Sanyo SCP-7050 | ARM9 | 1.004SP | Sprint | J2ME | 1537200 | 226 | 376456 |
| MWIPS | MWIPS Battery | Grinder Mark | HTTP Time | HTTP15 Battery | HTTP30 Battery | HTTP60 Battery | UDP Time |
| 158 | 442707 | 3355 | 4261.31 | 34405 | 59783 | 78771 | 1523.7 |
| UDP15 Battery | UDP30 Battery | UDP60 Battery | FixTime Clear | FixTime Obscured | EAU Clear | EAU Obscured | GPS15 Battery |
| 32507 | 61270 | 108089 | 1.77 | 8.5 | 24.94 | 81.63 | 52993 |
| GPS30 Battery | GPS60 Battery | GPSHTTP 15 Battery | GPSHTTP 30 Battery | GPSHTTP 60 Battery | GPSUDP 15 Battery | GPSUDP 30 Battery | GPSUDP 60 Battery |
| 47758 | 44390 | 43448 | 64518 | 80152 | 25501 | 37654 | 55193 |

## 5.7 Findings in the Application of the Framework

A number of observations about the particular model used as an example may be made from the values obtained by means of the battery of framework tests.

It is immediately obvious that any operations performed on a continuous basis dramatically reduce expected battery life. The baseline level, when the phone is performing no operations, provides an active duration of almost eighteen days. During this time the phone is free to function as indicated by its primary use, to send and receive phone calls. Naturally, any such activity will contribute to a decline in the battery life, but the phone is potentially able to receive calls for the full length of its activation. This is a key consideration for developers who wish to code applications that will actively reside on the phone. The application with the longest expected battery life within this framework is the processing of Whetstone calculations, and the phone shut down after a mere 5.12 days. The phone, in essence, lasts 3.4 times longer when doing nothing except waiting to receive phone calls than when performing these repeated calculations.

When running Dhrystone, the battery life predicted for the Sanyo SCP-7050 by this framework is 376, 456 seconds, or 4.36 days. This is a 14.84% reduction in expected lifetime. This may be explained by examining the number of operations performed by the phone for each of these benchmarks. More Dhrystone operations were completed, 226 million Dhrystone instructions per second, (DMIPS) than Whetstone, which was measured at 158 million Whetstone instructions per second (MWIPS). The framework therefore indicates that 30.01% fewer floating-point operations are performed by the processor of this particular phone model during that time, although as with all benchmarks this may be somewhat dependent upon the compiler and available libraries. In total, the number of integer operations as measured by the Dhrystone benchmark during the duration of the battery charge was 8.5x1013, while the battery life allowed 6.99x1013 total floating-point operations.

By way of evaluating the instruction-per-second values using an external standard, Table 5.2 provides the DMIPS and MWIPS results of some PC and mobile processors, as well as those of some peripheral processors, which are used in specialized electronic devices such as remote controls and other device controllers [88][89][90][91][92][93][94]. The framework's results on the Sanyo model's processor, ARM9 [94], have been included for ease of comparison. The information available online provides data that is conspicuous for its incompleteness regarding an assessment based upon both CPU benchmarks; this is largely due to the fact that until recently the software architecture for mobile phones (up to CLDC 1.0) did not support floating-point operations [25]. With more recent models and software support for these calculations, the need for consistent evaluation of newly released and future mobile devices is highlighted.

Table 5.2 – Sample DMIPS and MWIPS Values

| Processor | Type | DMIPS | MWIPS |
|---|---|---|---|
| ARM9 | Mobile | 226 | 158 |
| V850ES/Hx3 | Peripheral | 69 | |
| VR4305 | Peripheral | 105 | |
| ARM 926EJ-S core1 | Mobile | 300 | |
| ARM 7TDMI | Mobile | 142 | |
| Cortex M3 | Mobile | 169 | |
| ARM 968E-S | Mobile | 289 | |
| ARM 946E-S | Mobile | 231 | |
| Diamond 108Mini | Mobile | 335 | |
| AMD K6 | Notebook | 555 | 317 |
| Intel Pentium | PC | 245 | 145 |
| Intel Celeron 533 | PC | 1822 | 708 |
| Intel Celeron 366 | PC | 1249 | 485 |

The GrinderMark measure, while relatively new, provides a basis for comparison when the framework's result in this aspect is contrasted with other, similar devices that have their values reported on the EEMBC website [62]. A sample of these is displayed in Table 5.3. The Sanyo SCP-7050 performs better than all the Sony Ericsson and Motorola models, and all but one of the Nokia phones that have scores reported there. It scores lower, however, than the Audiovox, Benq-Siemens, Toshiba and iMate entries in the list.

Table 5.3 – Sample GrinderMark Values

| Phone Model | GrinderMark |
|---|---|
| Sanyo SCP-7050 | 3355 |
| Motorola Razr V3 | 131 |
| Motorola V600 | 140 |
| Sony Ericsson k750i | 2642 |
| Sony Ericsson S700i | 1750 |
| Nokia 6230 | 1458 |
| Nokia 6600 | 3830 |
| Audiovox SMT 5600 | 4876 |
| Benq-Siemens E71 | 6773 |
| Toshiba 902T, TS803 | 5356 |
| iMate SPL | 8631 |

Since the total UDP transmission time (*i.e.*, the time it takes for the client to send and the server to receive the data) of the Sanyo was consistently less than a second in numerous trials, the synchronicity problem outlined in Chapter 4 rendered a precise value impractical to attain in a framework intended for general use. The client-side transmission time, however was measured at about a second and a half (1523.7 ms). A comparison of the wireless protocols confirms what might be expected of the difference between UDP and HTTP transmission times; the RESTful HTTP transmission period, including the time taken for the client to receive an acknowledgement message, was measured at 4261.31 ms, 2.8 times longer than its one-way UDP counterpart.

The battery life expectation while performing these functions repeatedly at various intervals (Figure 5.5 and Figure 5.6) reveals a logical trend: the greater the interval between wireless transmissions, the less work the phone performs, and the longer the battery lasts. It is generally the case that, across the intervals, transmitting the same amount of information (*i.e.*, the String variable defined in Section 4.5) via HTTP is slightly more costly than by UDP. At 15 second intervals, the battery is depleted in 5.52% less time when performing HTTP transmissions. At the 30 and 60 second intervals, however, the battery is depleted in 2.43% and 27.12% (respectively) less time performing UDP transfers. The almost negligible time differences at the 15 and 30 second intervals may be accounted for by the short rest-time between wireless activities. Since these lower parameters produce almost continuous activation of the transmitting hardware, the greater differences in power consumption due solely to the activities themselves may only be seen at the higher intervals.

Fix times show a great susceptibility to environmental conditions. In relatively unobstructed areas GPS data can be obtained in less than two (1.77) seconds, while indoors each fix requires almost ten (8.5); the indoor fixes take 380.23% longer than the outdoor ones to obtain. The Estimated Accuracy Uncertainty is also dramatically affected, revealing a difference of 24.94 meters vs. 81.63m for outdoor vs. indoor respectively. These figures indicate a 227.3% greater confidence in the accuracy of position information obtained in a clear area (note that a lower accuracy uncertainty is a higher degree of confidence). These results place a quantitative value on the results that one might expect: in the absence of environmental obstructions, GPS hardware is able to receive position information more quickly and with greater estimated accuracy than in more restricted circumstances. The framework reveals the magnitude of these differences within the conditions outlined in the experimental description.

The battery life recorded by the cellphone when performing GPS functions without transmitting any data appears, at first, to be counter-intuitive. When the interval between attempts to obtain GPS data is increased, the battery life actually appears to decline. It would stand to reason that in applications that allow more time to pass between the activation of the internal GPS hardware, the overall power consumption would be lower. It appears, however, that at lower interval settings the continuous activation of the GPS chip plays a greater role in determining the battery life than the intervals provided.

We note the values of the tests in microseconds show no clear trend, and the percentage differences in time are relatively small. Receiving GPS fixes at 30 second intervals results in a 9.88% reduction in battery life over the 15 second parameter. The 60 second setting, likewise, results in a 7.05% reduction in battery life over the 30 second

interval. No great difference is seen compared to the clear pattern that emerges when even higher intervals than a minute are used.

By way of investigating this effect, Figure 5.10 shows the result of running GPS-only tests with intervals of 300s, 600s, 1800s and 3600s in addition to the intervals used in this framework. A clear trend emerges according to the reasonable supposition, indicating that, due to the effect of continuous activation of the GPS chip, battery life can only be observed to decline in a predictable way for location-aware applications of this model at intervals exceeding one minute. A 4 second interval, which represents an even more clear case of continuous GPS activation, shows no decrease in battery life beyond the 15, 30 and 60 second intervals, providing further evidence for the hypothesized reason underlying this effect.

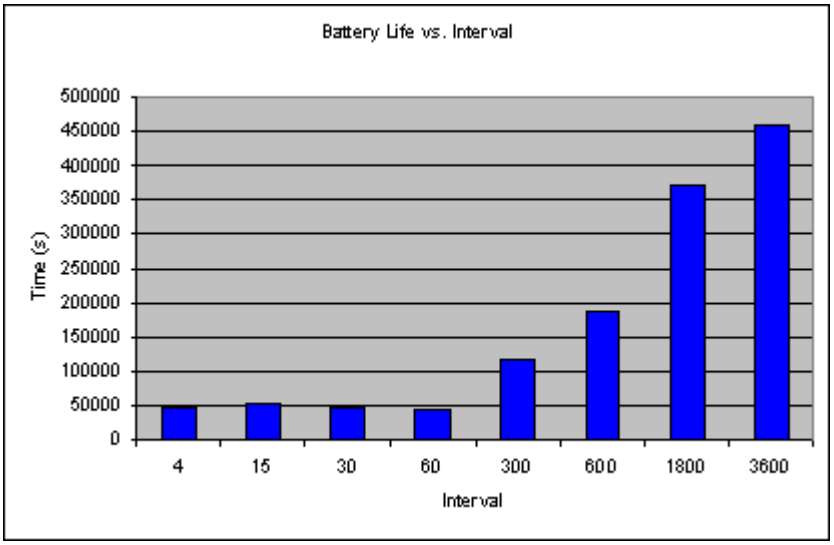


Figure 5.10 – Long-term GPS-only Testing

The framework tests demonstrate that below this interval value the phone can obtain frequent geographic updates without a noticeable change in the loss of battery power, an important factor for many tracking applications.

When the GPS reception functions are combined with the wireless transmission of data, this low-interval effect disappears. The energy expended in transmission procedures follows a predictable pattern that is not greatly affected by the activity of the GPS hardware except to reduce its overall time. In other words, while GPS functions do account for an increase in the decline of battery power, the difference this makes at request intervals of a minute or less does not make any significant impact on the trend observed by increasing the transmission intervals.

For transmissions of satellite data over RESTful HTTP, a 30 second delay between periods of receiving and transmitting geographical information yields a longer battery life by 48.49% over the 15 second delay. The 60 second setting provides an increase of 24.23% greater battery life over the 30 second version. The same steadily increasing battery life is observed over the intervals of receiving and transmitting data via UDP. The 30 second delay yields a longer battery life by 47.66% over the 15 second delay, and increasing the interval to 60 seconds provides an increase in duration of 46.58%.

It is interesting to note that some of the results obtained when both receiving GPS coordinates and transmitting received data indicate that the phone actually has a slightly longer expected battery life with the heavier workload. For example, the GPSHTTP60 Battery value is greater than the HTTP60 Battery value by 1.75%. It is theorized that since these activities both require significant resources, they actually prevent each other from functioning efficiently, and the cellphone spends more time processing the function

calls than performing the operation. We note from the framework that continuous calculations (*i.e.*, the DMIPS Battery and MWIPS Battery values) are not as costly as either GPS or wireless transmissions to perform. Further testing, beyond the scope of this framework, is required to confirm this, and may involve the software implementation on the phone's hardware that is manufacturer specific, and not open to the examination of third-party developers or researchers.

Six major recommendations that may be reasonably derived from the data obtained by the application of this framework are presented in Table 5.4.

Table 5.4 – Applications of Framework Data

| Observations | Impact on Design Decisions |
|---|---|
| Battery life with applications | Developers must be conservative with applications that perform continuous functions; both transmission and GPS functions more costly than CPU calculations |
| $8.5 \times 10^{13}$ integer vs. $6.99 \times 10^{13}$ floating-point operations during battery life | Far fewer floating point operations performed during battery life duration, so developers should use integer operations when it is possible to do so |
| HTTP transmission time 2.8 times longer than UDP time | Real-time applications should use UDP, while secure, vital transmissions require HTTP-type method |
| 380.23% longer GPS fix times indoors; 227.3% more confidence in outdoor data | Environmental impact on phone's ability to receive adequately reliable data is a major consideration for location-aware applications |
| Low-interval battery life | Applications that obtain data at intervals less than a few minutes do not lose extra battery power by using small intervals; good to know for real-time tracking applications |
| GPS and transmission battery life | Since low-interval effect disappears when applications transmit data, developers must once again take intervals into account for location-aware applications |

# Chapter 6

# Conclusions and Directions for Future Research

## 6.1 Contributions of This Research

- A structured evaluation framework was developed to measure a number of conceptually linked performance metrics chosen according to the justifications provided in Chapters 3 and 4 of this dissertation. The results of applying it to a selected model of cellphone were reported in Chapter 5.
- The framework's techniques are easily reproduced and applied as long as a developer has the resources that would be required for any significant software development anyway, these being the cellphone itself, a server with access to the internet, and the software tools for creating and deploying web services and client-side software.
- The specific software tools required to run the tests described in this framework are easily created or adapted from existing and publicly available code (*e.g.*, the Dhrystone and Whetstone benchmarks). The GrinderBench software is available for download free of charge [61].
- The application of the evaluation system described in this dissertation, applied to a single model of cellphone, yielded results that are described in Chapter 5. Comparisons between the result matrix produced for the Sanyo SCP-7050 and that obtained for other models will assuredly yield revealing figures.

## 6.2 Future Research

Whereas the framework presented in this dissertation represents a new conceptual model of the cellphone as an interconnected set of functional units, and approaches the determination of its performance along the lines of this model, it should not be assumed that this is necessarily the optimum system of evaluation for these devices. The metrics developed from this viewpoint provide a reasonable overview of the capabilities of the models tested based upon the current state of the research, and the needs with which this author is familiar; however, a number of performance measures might be suggested that were not included, and some that were included may prove less useful in practice than originally thought.

Future research may be done both with this framework, by applying it to new hardware as it becomes available in order to establish a data set for comparison, and upon the framework itself, refining and expanding its performance metrics and the information presented thereby. It is envisioned that as this system is utilized across various hardware models, and down through generations of phones, deficiencies in the metrics and procedures may come to light, and improvements made. On a "component" level, new metrics may replace old ones, or the matrix of values may be expanded to incorporate pertinent data. On a "metric" level, improved tools may be proposed to allow the more accurate determination of performance, or testing may be administered under conditions that provide a more useful basis for evaluation and comparison.

While the evaluation methods proposed alongside this framework are designed to provide useful and reasonably timely data for benchmarking phones as they become available to developers, it is also the case that if any intensive performance studies are to

be attempted using this conceptual model as a guide, a more rigorous system of reporting the obtained values will be required. The number of tests-per-metric would need to be increased in order to allow for statistical analyses, and for the development of confidence intervals. This will prove to be particularly challenging where wireless transmission of data is concerned, as the reliability of the network upon which these transmissions depend can fluctuate based upon a number of factors such as location, time, weather conditions, and data traffic.

One area of potential interest for application developers as they apply these metrics to new hardware is the rate at which the different functions of cellular phones improve with time, and how uniformly these changes take place. For example, will GPS accuracy improve dramatically in the next few years? Will CPU cycle time increase to the degree that processor performance will be noticeably affected? Will they develop in tandem with one another, or will one metric change more rapidly? How will these changes affect power consumption?

It is anticipated that the contributions of this proposed framework will be validated by its use, particularly in terms of its overall approach to the problem of obtaining performance measures. At the same time, developments in cellphones and their communication methods may produce methods and functions rendering the current specifics of functionality that this framework examines irrelevant or of reduced usefulness. If the application interfaces develop in such a way that developers begin to use methods other than UDP and HTTP in common practice, any reasonable evaluation method should take this shift in usage into account, and its performance metrics adjusted accordingly.

## References


[1] "Up to 90 percent of globe to have mobile coverage," *textually.org*, October 17, 2006, retrieved on April 12, 2007,
URL: http://www.textually.org/textually/archives/2006/10/013841.htm

[2] B. Charney, "Lowering the bar for wireless Web," *CNET News.com*, August 21, 2002, retrieved on April 12, 2007,
URL: http://news.com.com/2100-1033-954614.html

[3] K. Maney, "Baby's arrival inspires birth of cellphone camera – and societal evolution," *USA Today*, January 23, 2007, retrieved on April 12, 2007,
URL: http://www.usatoday.com/tech/columnist/kevinmaney/2007-01-23-kahn-cellphone-camera_x.htm

[4] W. Strauss, "Cellphones suck up the chips," *EETimes Online*, January 5, 2004, retrieved on April 12, 2007,
URL: http://www.eetimes.com/op/showArticle.jhtml?articleID=18310801

[5] MOTODEV, The Motorola developer network,
URL: http://developer.motorola.com

[6] E. A. Powell, "Citi Introduces Cellphone Banking as 'A Natural Extension,'" *CRM News*: Finance, April 5, 2007, retrieved on April 12, 2007,
URL: http://www.crmbuyer.com/story/56715.html

[7] A. Kotok, "This Time, We Mean It," August 6, 2004, retrieved on April 12, 2007, *ScienceCareers.org*, Career Development: Articles, URL: http://sciencecareers.sciencemag.org/career_development/previous_issues/articles/3150/this_time_we_mean_it

[8] A. Steinhage (2005), "Advanced User-Interface Technologies for Mobile Applications," *Proceedings of the 13th Wireless World Research Forum*, Jeju Island, Korea, March 2005, URL: http://www.future-shape.de/publications_steinhage/WWRF2005UserInterfaces.pdf

[9] S. Barbeau, M. Labrador, P. Winters, R. Perez, N. Georggi, "A General Architecture in Support of Interactive, Multimedia, Location-Based Mobile Applications," *IEEE Communications*, Institute of Electrical and Electronics Engineers, Inc., November 2006, Vol. 44, No. 11. pp. 156-163

[10] R. Meier, “Communication paradigms for mobile computing,” *ACM SIGMOBILE Mobile Computing and Communications Review*, October 2002, Volume 6, Issue 4, pp. 56 - 58

[11] J. Flinn, “Extending Mobile Computer Battery Life through Energy-Aware Adaptation,” Ph.D. Thesis, December 2001, Carnegie Mellon University, Pittsburgh, PA

[12] T. Starner, Y. Maguire, “Heat dissipation in wearable computers aided by thermal coupling with the user,” *Mobile Networks and Applications*, March 1999, Volume 4, Issue 1, pp. 3 – 13

[13] *iDEN J2ME™ Developer’s Guide 2005*, Version 1.98, © Motorola, Inc. 2005

[14] S. W. Gibson (1997), *Cellular Telephones and Pagers: An Overview*, pp. 11-13, Butterworth-Heinemann

[15] J. L. Hennessey, D. A. Patterson (1998), *Computer Organization and Design: The Hardware / Software Interface*, p. 21, Morgan Kaufmann Publishers, Inc.

[16] T. N. Trainor, D. Krasnewich (1996), *Computers!, Fifth Edition*, p. 98, The McGraw-Hill Companies, Inc.

[17] M. van Steen, P. Homburg, A. S. Tanenbaum (1999), “Globe: A Wide-Area Distributed System,” *IEEE Concurrency*, vol. 7, pp. 70-78, Jan-March 1999

[18] “Making History: Developing the Portable Cellular System,” motorola.com, retrieved on May 27, 2007,
URL: http://www.motorola.com/content.jsp?globalObjectId=7662

[19] S. Mehta, “Building a cellphone for the masses,” December 1, 2006, retrieved on May 24, 2007, Fortune Magazine, as published on CNNMoney.com, URL: http://money.cnn.com/2006/12/01/technology/personaltech/plugged.in.motorola.fortune/index.htm

[20] “Mobiledia: New Cellphone Releases,” retrieved on May 27, 2007,
URL: http://www.mobiledia.com/rss/phones/new/

[21] B. Weinberg (2006), “Get longer battery life in Linux-based handsets,” Mobile Handset Design Line, Retrieved on May 28, 2007, URL: http://www.mobilehandsetdesignline.com/showArticle.jhtml?articleID=177100471

[22] C. K. Ng, M. Savvides, P. K. Khosla, "Real-time face verification system on a cell-phone using advanced correlation filters," *Automatic Identification Advanced Technologies, 2005. Fourth IEEE Workshop*, pp. 57-62, October 17-18, 2005, URL: http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/10364/32967/01544401.pdf?arnumber=1544401

[23] DoJa *1.5 Overseas Edition DoJa Java Content Developer's Guide*, Release 1.2, p. 36, November 22, 2002, NTT DoCoMo,Inc.,
URL: http://www.doja-developer.net/downloads/downld.php?id=12

[24] J. H. Day (2002), "Mobile Devices: Cell-phone display ICs eye colorful future," *EE Times*, January 3, 2002, retrieved on May 28, 2007,
URL: http://www.eetimes.com/showArticle.jhtml?articleID=16504200

[25] S. Li, J. Knudsen (2005), *Beginning J2ME: From Novice To Professional, Third Edition*, p. 305, Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013

[26] J. Cohen (2000), "Speech interface belongs right in mobile device," *EE Times*, November 27, 2000, retrieved on May 28, 2007,
URL: http://www.eetimes.com/op/showArticle.jhtml?articleID=18304954

[27] J. Swartz (2007), "Vista's coming, so how should PC users handle it?" USA Today, January 25, 2007, retrieved on May 28, 2007,
URL: http://www.usatoday.com/tech/products/2007-01-21-handling-vista_x.htm

[28] S. Li, J. Knudsen (2005), *Beginning J2ME: From Novice To Professional, Third Edition*, p. 4, Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013

[29] A. Küpper (2005), *Location-based Services: Fundamentals and Operation*, p. 91, John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex PO 19 8SQ, England

[30] V. Livingston (2006), "Two Billion GSM Customers Worldwide," *PR Newswire*, June 13, 2006, retrieved on May 20, 2007,
URL: http://www.prnewswire.com/cgi-bin/stories.pl?ACCT=109&STORY=/www/story/06-13-2006/0004379206&EDATE=

[31] Küpper, p. 93

[32] Küpper, p. 98

[33] Küpper, p. 99-101

[34] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee (1999), "Hypertext Transfer Protocol -- HTTP/1.1," Network Working Group, Category: Standards Track, The World Wide Web Consortium (W3C), retrieved on April 22, 2007, URL: http://www.w3.org/Protocols/HTTP/1.1/rfc2616.pdf

[35] *ibid.*, Section 9: Method Definitions

[36] "Interface Servlet," *JavaTM 2 Platform Enterprise Edition, v 1.4 API Specification*, 2003, retrieved on April 22, 2007, URL: http://java.sun.com/j2ee/1.4/docs/api/javax/servlet/Servlet.html

[37] "Connectionless Transport: UDP," The Computer Networks Research Group, University of Massachusetts, Amherst, 1996-2000, retrieved on April 23, 2007, URL: http://www-net.cs.umass.edu/kurose/transport/UDP.html

[38] J. Postel (Editor) (1981), "Transmission Control Protocol: DARPA Internet Program Protocol Specification," prepared for the Defense Advanced Research Projects Agency by the Information Sciences Institute, University of Southern California, 4676 Admiralty Way, Marina del Rey, California 90291, pp. 1 - 4, retrieved on April 26, 2007, URL: http://tools.ietf.org/html/rfc793

[39] *ibid*., pp. 21 - 22

[40] C.M. Kozierok (2005), "TCP Connection Termination," *The TCP/IP Guide*, © 2003-2005 Charles M. Kozierok, retrieved on April 23, 2007, URL: http://www.tcpipguide.com/free/t_TCPConnectionTermination-2.htm

[41] JSR-101 Expert Group, *Java™ API for XML-based RPC, JAX-RPC 1.1*, JSR-101 JSR-101 Expert Group, Java Community Process (JCP), Maintenance Release Version 1.1, p. 11

[42] A. Chowdhury, P. Choudhary (2002), *JAX: Java APIs for XML Kick Start*, Chapter 11: "Working with JAX-RPC," Sams Publishing, ISBN:0-672-32434-2, reproduced on developer.com, retrieved on April 23, 2007, URL: http://www.developer.com/java/ent/article.php/2109561

[43] "Web Services Activity Statement," prepared for the May 2007 W3C Advisory Committee Meeting, Copyright © 2007 W3C® (MIT, ERCIM, Keio), retrieved on April 23, 2007, URL: http://www.w3.org/2002/ws/Activity

[44] “Enhancing Transportation Safety and Security via Scalable Location-Based Wireless Applications,” prepared for the June 2006 University Consortium on Intermodal Transportation Safety and Security by the Center for Urban Transportation Research, University of South Florida, 4202 East Fowler Ave., CUT100, Tampa, FL 33620, Project Number: FTA-FL-26-7102-01

[45] S. Barbeau, M. Labrador, P. Winters, R. Pérez, N. Georggi (2006), “A General Architecture in Support of Interactive, Multimedia, Location-Based Mobile Applications,” *IEEE Communications Magazine*, pp. 156-163, November 2006

[46] “Enhanced 911 - Wireless Services,” Federal Communication Commission 911 services website, retrieved on June 1, 2007,
URL: http://www.fcc.gov/911/enhanced/

[47] “What You Need To Know About Calling 911 From Your Wireless Phone,” FCC Consumer Advisory, December 14, 2006, retrieved on June 1, 2007,
URL: http://www.fcc.gov/cgb/consumerfacts/e911.html

[48] C. Needles (2006), “Tele Atlas Showcases Wireless Location-Based Applications at CTIA Wireless,” Directions Magazine, September 12, 2006, retrieved on June 1, 2007,
URL:
http://www.directionsmag.com/press.releases/index.php?duty=Show&id=15263&trv=1

[49] G. Letham (2006), “The 2007 Navteq LBS challenge... Putting Location-Based services On The Map!” The Location Based Service LBS Zone website, April 06, 2007, retrieved on June 1, 2007,
URL: http://www.lbszone.com/content/view/1814/45/

[50] P.J. Kiviat (1974), “A Challenge to Benchmarking,” *Benchmarking: Computer Evaluation and Measurement*, Appendix E, p. 177, Hemisphere Publishing Corporation, 1025 Vermont Avenue, N.W., Washington, D.C., 20005

[51] R. Weicker (2001), “SPEC OSG Benchmarks: History, Use, and Current Uses,” *Performance Evaluation and Benchmarking with Realistic Applications*, p. 21, The MIT Press, Cambridge, Massachusetts

[52] I. Robinson (2004), “Inside Benchmarking: Hardware Testing in the 21st Century,” TechNewsWorld, January 16, 2004, retrieved on June 18, 2007,
URL: http://www.technewsworld.com/story/32608.html

[53] N. Benwell, (1974), “An Introduction to Benchmarking,” *Benchmarking: Computer Evaluation and Measurement*, pp. viii - ix, Hemisphere Publishing Corporation, 1025 Vermont Avenue, N.W., Washington, D.C., 20005

[54] *Benchmark Procurement Guidelines for Government PC Buyers*, Q1, 2006 Edition, p. 2, © 2002-2005 Advanced Micro Devices, Inc.

[55] P. Hatt (1974), "The Role of Benchmarking," *Benchmarking: Computer Evaluation and Measurement*, pp. 1-13, Hemisphere Publishing Corporation, 1025 Vermont Avenue, N.W., Washington, D.C., 20005

[56] V. Viswanath, J. A. Abraham, W. A. Hunt, Jr. (2005), "RTL Annotations for Low Power Microprocessor Design," ACEED 2005 Submission for PUBLIC session, URL: http://www.research.ibm.com/aceed/2005/posters/viswanath-abstract.pdf

[57] R. P. Weiker (1990), "An Overview of Common Benchmarks," *Computer*, Volume 23, Issue 12 (December 1990), IEEE Computer Society Press, Los Alamitos, CA, USA

[58] Linux Benchmark Suite Homepage, retrieved on June 18, 2007, URL: http://lbs.sourceforge.net/

[59] SPEC – Standard Performance Evaluation Corporation website, retrieved on June 18, 2007, URL: http://www.spec.org/

[60] Transaction Processing Performance Council website, retrieved on June 18, 2007, URL: http://www.tpc.org/

[61] EEMBC -- The Embedded Microprocessor Benchmark website, retrieved on June 22, 2007, URL: http://www.eembc.org/

[62] "GrinderBench Benchmark Scores" from the GrinderBench website, retrieved on June 22, 2007, URL: http://www.grinderbench.com/benchmarks/

[63] G. Chen, M. Kandemir, N. Vijaykrishnan, M.J. Irwin, M.J. (2002), PennBench: a benchmark suite for embedded Java, IEEE International Workshop on Workload Characterization, November 25, 2002, ISBN: 0-7803-7681-1, URL: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1226495

[64] F. Parienté (2001), "Performance Analysis and Monitoring Using Hardware Counters," article on the Sun Developer Network (SDN), December 2001, retrieved on July 4, 2007, URL: http://developers.sun.com/solaris/articles/hardware_counters.html

[65] J. Stokes, "Behind the benchmarks: SPEC, GFLOPS, MIPS et al," *Ars Technica* website, Copyright © 1998-2004, Ars Technica, LLC, retrieved on July 4, 2007, URL: http://arstechnica.com/cpu/2q99/benchmarking-1.html

[66] E. Strohmaier (2006), "Performance Complexity: An Execution Time Metric to Characterize the Transparency and Complexity of Performance," *CTWatch Quarterly*, Volume 2 Number 4B, November 2006, retrieved on July 15, 2007,
URL: http://www.ctwatch.org/quarterly/articles/2006/11/performance-complexity-an-execution-time-metric-to-characterize-the-transparency-and-complexity-of-performance/

[67] A. El-Rabbany (2002), *Introduction to GPS, The Global Positioning System*, Artech House, Inc., 685 Canton Street, Norwood, MA 02062

[68] JSR-179 Location API for J2ME v1.0.1,
URL: http://mobilezoo.biz/jsr/179/javax/microedition/location/package-summary.html

[69] S. Horsley (2007), "Tech Leaders Seek Computer Efficiency," National Public Radio (NPR), Technology, Morning Edition, June 13, 2007, retrieved on July 15, 2007,
URL: http://www.npr.org/templates/story/story.php?storyId=11007403

[70] A. Jossen, V. Spath, H. Dohring, J. Garche (1999), "Battery Management Systems (BMS) for Increasing Battery Life Time," *The 21st International Telecommunications Energy Conference*, June 1999, INTELEC '99

[71] A. R. Weiss (2002), *Dhrystone Benchmark: History, Analysis, "Scores" and Recommendations*, White Paper, EEMBC Certification Laboratories (ECL), LLC, 6507 Jester Blvd, Suite 511, Austin, Texas 78750

[72] Dhrystone Benchmark in Java, Computer Creators, Inc.,
URL: http://www.c-creators.co.jp/okayan/DhrystoneApplet/, retrieved on November 11, 2007

[73] Whetstone for Java, Hard Realtime Java for Safety Critical Systems, http://aicas.com/, URL: http://www.aicas.com/download/Whetstone.java, retrieved on November 13, 2007

[74] Float11 (ver 0.51),
URL: http://henson.newmail.ru/j2me/Float11.htm, retrieved on November 13, 2007

[75] R. T. Fielding (2000), "Architectural Styles and the Design of Network-based Software Architectures," Chapter 5: Representational State Transfer (REST), PhD Dissertation, University of California, Irvine,
URL: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[76] R. McMillan (2003), "A RESTful approach to Web services," *Network World*, February 17, 2003, retrieved on January 6, 2008,
URL: http://www.networkworld.com/ee/2003/eerest.html?page=1

[77] K. Monson (2007), "Are You Ready for Daylight Saving Time?" *PC Magazine*, May 5, 2007, retrieved on July 29, 2007,
URL: http://www.pcmag.com/article2/0,1895,2100064,00.asp

[78] E. Murakami, D. P. Wagner (1999), "Can using global positioning system (GPS) improve trip reporting?" *Transportation Research Part C: Emerging Technologies Volume 7*, Issues 2-3, April-June 1999, Pages 149-165,
URL: http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B6VGJ-3X9RYH9-6&_user=10&_rdoc=1&_fmt=&_orig=search&_sort=d&view=c&_acct=C000050221&_version=1&_urlVersion=0&_userid=10&md5=cccae4da47dbb5e42fa0926cd42cbbc6

[79] D. Aguilar, S. Barbeau, R. Perez, M. Labrador, P. Winters (2007), "A Comparison of Fix Times and Estimated Accuracies in Application Programming Interfaces (APIs) for GPS Enabled Mobile Phones," *Proceedings of the 11th World Conference on Transport Research*, Berkeley, CA, June 2007

[80] "Motorola Introduces Rugged i580 iDEN Phone," April 4, 2006, *Mobiledia*, retrieved on August 16, 2007, URL: http://www.mobiledia.com/news/45750.html

[81] Image of Motorola iDEN i580 taken from URL
www.letstalk.com/reviews/reviewdtl.htm?pfId=1114

[82] "Ericsson Reports Positive Cash Flow and Continued Progress in Cost Reductions," Press release, February 3, 2003, Ericsson website, retrieved on January 6, 2008,
URL: http://www.ericsson.com/ericsson/press/releases/20030203-890167.shtml

[83] B. O'Shaughnessy (1999), "The Move to CDMA: Bell Mobility's Technology Decision," June 1, 1999, CMDA Development Group website,
URL: http://www.cdg.org/resources/Guest_column.asp?columnid=18

[84] Image of Sanyo SCP-7050 taken from URL
http://www.letstalk.com/reviews/review.htm?pfId=1274

[85] Image of Sanyo SCP-7050 taken from URL
http://www.mycellphoneblog.com/sprints-rugged-sanyo-scp-7050/

[86] Sanyo SCP-7050 (Sprint) Cellphone Detail,
URL: http://www.letstalk.com/product/product.htm?prId=32654

[87] "MIPS Architecture Enabling Growing List of Mobile Application Processors," D&R Headline News, August 30, 2004, retrieved on February 10, 2008,
URL: http://www.us.design-reuse.com/news/8544/mips-architecture-enabling-list-mobile-processors.html

[88] “NEC Offers Two High Cost Performance 64-bit RISC Microprocessors,” NEC Press Release, January 20, 1988, retrieved on February 10, 2008,
URL: http://www.nec.co.jp/press/en/9801/2002.html

[89] “MCUs execute up to 69 dhrystone MIPS at 32 MHz clock speed,” NEC Press Release, October 5, 2007, retrieved on February 10, 2008,
URL: http://news.thomasnet.com/fullstory/804810

[90] G. Shvets (2007), “AMD Mobile K6 266 - AMD-K6/266ACZ,” CPU World Website, October 15, 2007, retrieved on February 10, 2008,
URL: http://www.cpu-world.com/CPUs/K6/AMD-Mobile%20K6%20266%20-%20AMD-K6-266ACZ.html

[91] “INTEL PENTIUM 133 mhz,” Product Description, DeRemate.com website, retrieved on February 10, 2008,
URL: http://oferta.deremate.com.ar/id=19083024_intel-pentium-133-mhz

[92] G. Shvets (2007), “Intel Celeron 533 - FV80524RX533128 (FV524RX533 128),” CPU World Website, November 25, 2007, retrieved on February 10, 2008,
URL: http://www.cpu-world.com/CPUs/Celeron/Intel-Celeron%20533%20-%20FV80524RX533128%20(FV524RX533%20128).html

[93] G. Shvets (2007), “Intel Celeron 366 - FV80524RX366128 (FV524RX366 128),”CPU World Website, November 25, 2007, retrieved on February 10, 2008,URL: http://www.cpu-world.com/CPUs/Celeron/Intel-Celeron%20533%20-%20FV80524RX533128%20(FV524RX533%20128).html

[94] S. Segan (2007), “Sanyo SCP-7050,” Cellphones at PCMag.com, March 26, 2007, retrieved on February 11, 2008,
URL: http://www.pcmag.com/article2/0,1759,2107862,00.asp

**Appendices**

## Appendix A: Code Segment for Evaluating Power Consumption

Baseline power consumption was measured by simply observing the phone as it naturally depleted its battery power, running no third-party applications and receiving no calls. The following code segment was attached to all the applications of this framework that performed repeated operations (*e.g.*, transmitting UDP data, running the MIPS benchmark) in order to detect the decline in battery power as it progressed through the different levels of battery strength.

The timestamps of changes in battery level were written to the cellphone's record store, and this persistent memory is read after the phone shuts down due to continued use and is subsequently recharged. The java.io and javax.microedition.rms libraries need to be imported into the application in order to access the record store.

It should be noted that the final timstamp cannot be measured within the software itself, since it is a record of the time at which the phone actually powers down due to a fully depeleted battery. In order to obtain this value, once the phone's battery level reaches its lowest reading, the time is recorded every few data transmissions. The number of transmissions between timestamp recording is determined by the variable lowBatteryInterval. In the framework represented by this dissertatoin that value was set to 4; thus, every fourth transmission after the battery level is determined to be "low" is timestamped and this value is written to the record store.

Naturally, this method introduces an uncertainty error into the battery duration. An examination of the battery decline charts, however (*e.g.*, Figure 5.6) reveals that the amount of time spent in "low" battery status is quite small (3.12% of the total time

**Appendix A (Continued)**

averaged over the three trials represented by Figure 5.6) and as long as the lowBatteryInterval is kept constant over all framework tests the comparative ability of the figures obtained is not compromised.

In the initialization procedure:

```
/* Initialize report variables */
this.sentPackets = 0;
this.batteryLevel = getBatteryLevel();

/* Record start time for insertion into the record store if first use */
startTime = java.lang.System.currentTimeMillis();

//Populate records with default values
openRecords();
try {
   if ((BTRMS.getNumRecords() == 0) || (BTRMS == null)) { //This is not the first use
      java.lang.System.out.println("Initializing Record Store.");
      String str = "N/A";
      byte[] rec = str.getBytes();
      for (int n = 0; n < MAXRECORDS; n++) {
         BTRMS.addRecord(rec, 0, rec.length);
      }
   }
} catch (Exception E) {
}
closeRecords();
```

In main thread:

```
while (true) {

        //Perform repeated function

        this.sentPackets++; //Keeps track of how many times the operation ran

        if (this.batteryLevel != this.get_Battery_Level()) {
                java.lang.System.out.println("Triggering record store functions.");
                if (this.batteryLevel == 3) {
                        this.openRecords(); // Record start time
                        Date tempDate = new Date(this.startTime);
                        this.insertRecord(getTimeString(tempDate),1);
```

**Appendix A (Continued)**

```
                        // Record time of decline from battery level 3 to level 2
                        tempDate = new Date(java.lang.System.currentTimeMillis());
                        this.insertRecord(getTimeString(tempDate),2);
                }
                if (this.batteryLevel == 2) {
                        this.openRecords();
                        Date tempDate = new Date(java.lang.System.currentTimeMillis());
                        this.insertRecord(getTimeString(tempDate),3);
                        // Record time of decline from battery level 2 to level 1
                }
                if (this.batteryLevel == 1) {
                        this.openRecords();
                        Date tempDate = new Date(java.lang.System.currentTimeMillis());
                        this.insertRecord(getTimeString(tempDate),4);
                        // Record time of decline from battery level 3 to level 2
                }

                java.lang.System.out.println("Packets sent: " + this.sentPackets);
                this.insertRecord(""+this.sentPackets+"/N/A",6);
                //Record no. of packets sent or operations run
                this.closeRecords();
                this.batteryLevel = this.get_Battery_Level();
        }
        if (this.batteryLevel == 0) { //Store data every few runs to estimate end time for battery
                lowBatteryTimer++;
                if (lowBatteryTimer == lowBatteryInterval) {
                        java.lang.System.out.println("Triggering record store functions.");
                        this.openRecords();
                        Date tempDate = new Date(java.lang.System.currentTimeMillis());
                        this.insertRecord(getTimeString(tempDate),5);
                        java.lang.System.out.println("Packets sent: " + this.sentPackets);
                        this.insertRecord(""+this.sentPackets+",GPS:
                        "+communicator.user_Interface.get_GPSCount(),6);
                        this.closeRecords();
                        lowBatteryTimer = 0;
                }
        }
        }
        Perform user-interface operations
}
```

Record store procedures:

```
/**
 * Method called to open the Record Store
 */
public void openRecords() {
   try {
      BTRMS = BTRMS.openRecordStore("BTRMS", true);
```

## Appendix A (Continued)

```
    } catch (Exception e) {
        java.lang.System.out.println("Error opening record store: " + e);
    }
}

/**
 * Method called to close the Record Store
 */
public void closeRecords() {
    try {
        BTRMS.closeRecordStore();
    } catch (Exception e) {
        java.lang.System.out.println("Error opening record store: " + e);
    }
}

/**
 * Method called to delete the Record Store
 */
public void deleteRecords() {
    try {
        BTRMS.deleteRecordStore("BTRMS");
    } catch (Exception e) {
        java.lang.System.out.println("Error deleting record store: " + e);
    }
}

/**
 * Method called to insert a record into a particular index point
 * @param str user data for the Record Store
 */
public void insertRecord(String str, int recIndex) {
    byte[] rec = str.getBytes();
    try {
        BTRMS.setRecord(recIndex, rec, 0, rec.length);
    } catch (Exception e) {
        java.lang.System.out.println("Error updating record store: " + e);
    }
}
```

## Appendix B: Code Segments for Evaluating CPU Speed

This first code segment, and its associated functions, [72] is used to measure integer operations. The application incorporating this code produces a figure representing Dhrystone runs per second. This figure is then divided by 1,757 [71] in order to obtain the DMIPS value recorded in the evaluation framework.

Global variables:

```
static Record_Type Record_Glob, Next_Record_Glob; //See below for class Record_Type
static int Int_Glob;
static boolean Bool_Glob;
static char Char_Glob_1, Char_Glob_2;
static int[] Array_Glob_1    = new int[128];
static int[][] Array_Glob_2    = new int[128][128];
static Record_Type First_Record    = new Record_Type(),
       Second_Record   = new Record_Type();
```

In the main thread:

```
public static final int Ident_1    = 0;
public static final int Ident_2    = 1;
public static final int Ident_3    = 2;
public static final int Ident_4    = 3;
public static final int Ident_5    = 4;

for (Run_Index = 1; Run_Index <= Number_Of_Runs; ++Run_Index) {

   Proc_5();
   Proc_4();
   Int_Loc_1 = 2;
   Int_Loc_2 = 3;

   String_Loc_2 = "DHRYSTONE PROGRAM, 2'ND STRING";

   Enum_Loc[0] = Ident_2;
   Bool_Glob = !Func_2(String_Loc_1, String_Loc_2);

   while (Int_Loc_1 < Int_Loc_2) {
      Int_Loc_3_Ref[0] = 5 * Int_Loc_1 - Int_Loc_2;
      Proc_7(Int_Loc_1, Int_Loc_2, Int_Loc_3_Ref);
      Int_Loc_1 += 1;
   }
```

**Appendix B (Continued)**

```
    Int_Loc_3 = Int_Loc_3_Ref[0];
    Proc_8(Array_Glob_1, Array_Glob_2, Int_Loc_1, Int_Loc_3);
    Proc_1(Record_Glob);

    for (Char_Index = 'A'; Char_Index <= Char_Glob_2; ++Char_Index) {
        if (Enum_Loc[0] == Func_1(Char_Index, 'C'))
            Proc_6(Ident_1, Enum_Loc);
    }

    Int_Loc_3 = Int_Loc_2 * Int_Loc_1;
    Int_Loc_2 = Int_Loc_3 / Int_Loc_1;
    Int_Loc_2 = 7 * (Int_Loc_3 - Int_Loc_2) - Int_Loc_1;

    Int_Loc_1_Ref[0] = Int_Loc_1;
    Proc_2(Int_Loc_1_Ref);
    Int_Loc_1 = Int_Loc_1_Ref[0];

}
//End operations
end_time = java.lang.System.currentTimeMillis();
total_time = end_time - begin_time;
java.lang.System.out.println("Total time was: " + total_time);
result = (Number_Of_Runs * 1000 / total_time);
java.lang.System.out.println("Result: " + result + " Dhrystone/sec.");
```

Associated procedures and functions:

```
void Proc_1(Record_Type Pointer_Par_Val) {

    Record_Type Next_Record = Pointer_Par_Val.Record_Comp;
    Pointer_Par_Val.Record_Comp = Record_Glob;
    Pointer_Par_Val.Int_Comp = 5;
    Next_Record.Int_Comp = Pointer_Par_Val.Int_Comp;
    Next_Record.Record_Comp = Pointer_Par_Val.Record_Comp;
    Proc_3(Next_Record.Record_Comp);

    int[] Int_Ref = new int[1];

    if (Next_Record.Discr == Ident_1) {
        Next_Record.Int_Comp = 6;
        Int_Ref[0] = Next_Record.Enum_Comp;
        Proc_6(Pointer_Par_Val.Enum_Comp, Int_Ref);
        Next_Record.Enum_Comp = Int_Ref[0];
        Next_Record.Record_Comp = Record_Glob.Record_Comp;
        Int_Ref[0] = Next_Record.Int_Comp;
        Proc_7(Next_Record.Int_Comp, 10, Int_Ref);
        Next_Record.Int_Comp = Int_Ref[0];
    } else
        Pointer_Par_Val = Pointer_Par_Val.Record_Comp;
}
```

## Appendix B (Continued)

```
void Proc_2(int Int_Par_Ref[]) {

   int Int_Loc;
   int Enum_Loc;

   Int_Loc = Int_Par_Ref[0] + 10;
   Enum_Loc = 0;

   do
      if (Char_Glob_1 == 'A') {
         Int_Loc -= 1;
         Int_Par_Ref[0] = Int_Loc - Int_Glob;
         Enum_Loc = Ident_1;
      }
   while (Enum_Loc != Ident_1);
}
void Proc_3(Record_Type Pointer_Par_Ref) {

   if (Record_Glob != null)
      Pointer_Par_Ref = Record_Glob.Record_Comp;
   else
      Int_Glob = 100;

   int[] Int_Comp_Ref = new int[1];
   Int_Comp_Ref[0] = Record_Glob.Int_Comp;
   Proc_7(10, Int_Glob, Int_Comp_Ref);
   Record_Glob.Int_Comp = Int_Comp_Ref[0];
}

void Proc_4() {

   boolean Bool_Loc;

   Bool_Loc = Char_Glob_1 == 'A';
   Bool_Loc = Bool_Loc || Bool_Glob;
   Char_Glob_2 = 'B';

}

void Proc_5() {

   Char_Glob_1 = 'A';
   Bool_Glob = false;

}

void Proc_6(int Enum_Par_Val, int Enum_Par_Ref[]) {

   Enum_Par_Ref[0] = Enum_Par_Val;
```

## Appendix B (Continued)

```
   if (!Func_3(Enum_Par_Val))
      Enum_Par_Ref[0] = Ident_4;

   switch (Enum_Par_Val) {

   case Ident_1:
      Enum_Par_Ref[0] = Ident_1;
      break;

   case Ident_2:
      if (Int_Glob > 100)
         Enum_Par_Ref[0] = Ident_1;
      else
         Enum_Par_Ref[0] = Ident_4;
      break;

   case Ident_3:
      Enum_Par_Ref[0] = Ident_2;
      break;

   case Ident_4:
      break;

   case Ident_5:
      Enum_Par_Ref[0] = Ident_3;
      break;

   }
}

void Proc_7(int Int_Par_Val1, int Int_Par_Val2, int Int_Par_Ref[]) {

   int Int_Loc;

   Int_Loc       = Int_Par_Val1 + 2;
   Int_Par_Ref[0] = Int_Par_Val2 + Int_Loc;

}

void Proc_8(int[] Array_Par_1_Ref, int[][] Array_Par_2_Ref, int Int_Par_Val_1, int Int_Par_Val_2)
{

   int Int_Index,
      Int_Loc;

   Int_Loc = Int_Par_Val_1 + 5;
   Array_Par_1_Ref[Int_Loc] = Int_Par_Val_2;
   Array_Par_1_Ref[Int_Loc+1] = Array_Par_1_Ref[Int_Loc];
   Array_Par_1_Ref[Int_Loc+30] = Int_Loc;
   for (Int_Index = Int_Loc; Int_Index <= Int_Loc+1; ++Int_Index)
```

## Appendix B (Continued)

```
    Array_Par_2_Ref[Int_Loc][Int_Index] = Int_Loc;
    Array_Par_2_Ref[Int_Loc][Int_Loc-1] += 1;
    Array_Par_2_Ref[Int_Loc+20][Int_Loc] = Array_Par_1_Ref[Int_Loc];
    Int_Glob = 5;
}

int Func_1(char Char_Par_1_Val, char Char_Par_2_Val) {

    char Char_Loc_1, Char_Loc_2;

    Char_Loc_1 = Char_Par_1_Val;
    Char_Loc_2 = Char_Loc_1;
    if (Char_Loc_2 != Char_Par_2_Val)
        return Ident_1;
    else
        return Ident_2;

}

boolean Func_2(String String_Par_1_Ref, String String_Par_2_Ref) {

    int Int_Loc;
    char Char_Loc = '\0';

    Int_Loc = 2;
    while (Int_Loc <= 2)
        if  (Func_1(String_Par_1_Ref.charAt(Int_Loc),  String_Par_2_Ref.charAt(Int_Loc  +  1))  ==
Ident_1) {
            Char_Loc = 'A';
            Int_Loc += 1;
        }
    if (Char_Loc >= 'W' && Char_Loc < 'Z')
        Int_Loc = 7;
    if (Char_Loc == 'X')
        return true;
    else {
        if (String_Par_1_Ref.compareTo(String_Par_2_Ref) > 0) {
            Int_Loc += 7;
            return true;
        } else
            return false;
    }
}

boolean Func_3(int Enum_Par_Val) {

    int Enum_Loc;
```

**Appendix B (Continued)**

```
    Enum_Loc = Enum_Par_Val;
    if (Enum_Loc == Ident_3)
        return true;
    else
        return false;
}
```

Specialized class:

```
public class Record_Type {

    Record_Type Record_Comp;
    int Discr;
    int Enum_Comp;
    int Int_Comp;
    String String_Comp;
    int Enum_Comp_2;
    String String_Comp_2;
    char Char_Comp_1;
    char Char_Comp_2;
}
```

This second code segment and its associated procedures [72] is used to measure floating-point operations in the millions. In order to carry out certain operations not found in standard J2ME, the class *Float11.java* was imported into the application [73]. The result is a measure called KWIPS, or Kilo-Whetstone Instructions Per Second. This value, divided by $10^3$, provides the *MWIPS* (Millions of Whetstone Instructions Per Second) score that is recorded in the Framework.

Class variables:

```
static int ITERATIONS;
static int numberOfCycles;
static int cycleNo;
static double x1, x2, x3, x4, x, y, z[]=new double[1], t, t1, t2;
static double e1[]= new double[4];
static int i, j, k, l, n1, n2, n3, n4, n6, n7, n8, n9, n10, n11;
```

**Appendix B (Continued)**

In the main thread:

```
ITERATIONS = 10;
numberOfCycles = 100;
int numberOfRuns = 10;
float elapsedTime = 0;
float meanTime = 0;
float rating = 0;
float meanRating = 0;
int intRating = 0;
for (int runNumber=1; runNumber <= numberOfRuns; runNumber++) {
   java.lang.System.out.println(runNumber+ ". Test");

   elapsedTime = (float)(mainCalc()/1000);
// sum time in milliseconds per cycle
   meanTime = meanTime + (elapsedTime * 1000 / numberOfCycles);
// Calculate the Whetstone rating based on the time for
// the numbers of cycles just executed
   rating = (1000 * numberOfCycles) / elapsedTime;
// Sum Whetstone rating
   meanRating = meanRating + rating;
   intRating = (int)rating;
// Reset no_of_cycles for the next run using ten cycles more
   //Comment out below for time run
   numberOfCycles += 10;
   outputString += runNumber + ": " + rating + "\n";
   //Comment out above for time run
   LLinfo.update_SegmentID(outputString); //Used to show score
}

//End operations
meanTime = meanTime/numberOfRuns;
meanRating = meanRating/numberOfRuns;
intRating = (int)meanRating;
java.lang.System.out.println("Number of Runs " + numberOfRuns);
java.lang.System.out.println("Average time per cycle " + meanTime + " millisec.");
java.lang.System.out.println("Average Whetstone Rating " + intRating + " KWIPS");
outputString += "T: " + intRating; //Comment this out for time run and change below to "intRating"
LLinfo.update_SegmentID(outputString); //Used to show score
```

Whetstone procedures:

```
public static void p0() {
   e1[j] = e1[k];
   e1[k] = e1[l];
   e1[l] = e1[j];
}
```

## Appendix B (Continued)

```
public static void p3(double x,double y,double z[]) {
   x = t * (x + y);
   y = t * (x + y);
   z[0] = (x + y) /t2;
}

public static void pa(double e[]) {
   int j;
   j = 0;
   do {
      e[0] = ( e[0] + e[1] + e[2] - e[3] ) * t;
      e[1] = ( e[0] + e[1] - e[2] + e[3] ) * t;
      e[2] = ( e[0] - e[1] + e[2] + e[3] ) * t;
      e[3] = ( -e[0] + e[1] + e[2] + e[3] ) / t2;
      j += 1;}
   while (j < 6);
}

public static double mainCalc() { /* initialize constants */
   t = 0.499975;
   t1 = 0.50025;
   t2 = 2.0;
   /* set values of module weights */
   n1 = 0 * ITERATIONS;
   n2 = 12 * ITERATIONS;
   n3 = 14 * ITERATIONS;
   n4 = 345 * ITERATIONS;
   n6 = 210 * ITERATIONS;
   n7 = 32 * ITERATIONS;
   n8 = 899 * ITERATIONS;
   n9 = 616 * ITERATIONS;
   n10 = 0 * ITERATIONS;
   n11 = 93 * ITERATIONS;
   begin_time = java.lang.System.currentTimeMillis();
   for (cycleNo=1; cycleNo <= numberOfCycles; cycleNo++) {
      /* MODULE 1: simple identifiers */
      x1 = 1.0;
      x2 = x3 = x4 = -1.0;
      for(i = 1; i <= n1; i += 1) {
         x1 = ( x1 + x2 + x3 - x4 ) * t;
         x2 = ( x1 + x2 - x3 + x4 ) * t; // correction: x2 = ( x1 + x2 - x3 - x4 ) * t;
         x3 = ( x1 - x2 + x3 + x4 ) * t; // correction: x3 = ( x1 - x2 + x3 + x4 ) * t;
         x4 = (-x1 + x2 + x3 + x4 ) * t;
      }
// if (cycleNo==numberOfCycles) pout(n1, n1, n1, x1, x2, x3, x4);

      /* MODULE 2: array elements */
      e1[0] = 1.0;
      e1[1] = e1[2] = e1[3] = -1.0;
      for (i = 1; i <= n2; i +=1) {
```

**Appendix B (Continued)**

```
            e1[0] = ( e1[0] + e1[1] + e1[2] - e1[3] ) * t;
            e1[1] = ( e1[0] + e1[1] - e1[2] + e1[3] ) * t;
            e1[2] = ( e1[0] - e1[1] + e1[2] + e1[3] ) * t;



         e1[3] = (-e1[0] + e1[1] + e1[2] + e1[3] ) * t;
        }
// if (cycleNo==numberOfCycles) pout(n2, n3, n2, e1[0], e1[1], e1[2], e1[3]);
        /* MODULE 3: array as parameter */
        for (i = 1; i <= n3; i += 1)
            pa(e1);
// if (cycleNo==numberOfCycles) pout(n3, n2, n2, e1[0], e1[1], e1[2], e1[3]);
        /* MODULE 4: conditional jumps */
        j = 1;
        for (i = 1; i <= n4; i += 1) {
            if (j == 1)
                j = 2;
            else
                j = 3;
            if (j > 2)
                j = 0;
            else
                j = 1;
            if (j < 1 )
                j = 1;
            else
                j = 0;
        }
// if (cycleNo==numberOfCycles) pout(n4, j, j, x1, x2, x3, x4);
        /* MODULE 5: omitted */
        /* MODULE 6: integer arithmetic */
        j = 1;
        k = 2;
        l = 3;
        for (i = 1; i <= n6; i += 1) {
            j = j * (k - j) * (l -k);
            k = l * k - (l - j) * k;
            l = (l - k) * (k + j);
            e1[l - 2] = j + k + l; /* C arrays are zero based */
            e1[k - 2] = j * k * l;
        }
// if (cycleNo==numberOfCycles) pout(n6, j, k, e1[0], e1[1], e1[2], e1[3]);
        /* MODULE 7: trig. functions */
        x = y = 0.5;
        for(i = 1; i <= n7; i +=1) {
            x = t * Float11.atan(t2*Math.sin(x)*Math.cos(x)/(Math.cos(x+y)+Math.cos(x-y)-1.0));
            y = t * Float11.atan(t2*Math.sin(y)*Math.cos(y)/(Math.cos(x+y)+Math.cos(x-y)-1.0));
        }
// if (cycleNo==numberOfCycles) pout(n7, j, k, x, x, y, y);
```

## Appendix B (Continued)

```
        /* MODULE 8: procedure calls */
        x = y = z[0] = 1.0;
        for (i = 1; i <= n8; i +=1)
            p3(x, y, z);
// if (cycleNo==numberOfCycles) pout(n8, j, k, x, y, z[0], z[0]);
        /* MODULE9: array references */

        j = 0;
        k = 1;
        l = 2;
        e1[0] = 1.0;
        e1[1] = 2.0;
        e1[2] = 3.0;
        for(i = 1; i <= n9; i++)
            p0();
// if (cycleNo==numberOfCycles) pout(n9, j, k, e1[0], e1[1], e1[2], e1[3]);
        /* MODULE10: integer arithmetic */
        j = 2;
        k = 3;
        for(i = 1; i <= n10; i +=1) {
            j = j + k;
            k = j + k;
            j = k - j;
            k = k - j - j;
        }
// if (cycleNo==numberOfCycles) pout(n10, j, k, x1, x2, x3, x4);
        /* MODULE11: standard functions */
        x = 0.75;
        for(i = 1; i <= n11; i +=1)
            x = Math.sqrt( Float11.exp( Float11.log(x) / t1));
// if (cycleNo==numberOfCycles) pout(n11, j, k, x, x, x, x);
    } /* for */
    end_time = java.lang.System.currentTimeMillis();
    java.lang.System.out.println(" (time for " +numberOfCycles+ " cycles): "
            +(end_time - begin_time)+ " millisec.");
    return (end_time - begin_time);
}
```

## Appendix C: Code Segments for Evaluating Communication Time

This code segment is used to evaluate the phone-side UDP transfer time. It should be noted that since no response is sent from the corresponding server as a part of the protocol itself, it is not actually necessary to provide any server-side software support for this application, or even provide a valid server name and UDP port. Testing the transfer may be done, as in this case, by implementing a UDP listener that prints the transferred message to a computer screen monitoring server activity or a log file.

```
try {
   timeStampA = java.lang.System.currentTimeMillis();
   dc = (UDPDatagramConnection) Connector.open("datagram://" + serverName+":"+ udpPort);
   byte[] data = this.packageLocationData();
   Datagram dg = dc.newDatagram(data, data.length);
   dc.send(dg);
   dc.close();
   timeDiff = (java.lang.System.currentTimeMillis() - timeStampA);
   return timeDiff;
} catch (java.io.IOException ioException) {
   java.lang.System.err.println("Error sending UDP datagram: " + ioException);
   return -1;
} catch (Exception e) {
   java.lang.System.err.println("General error in UDP transmission function: " + e);
   return -1;
}
```

This function, referenced above, packages the String data as an array of bytes that is transmitted over the protocol.

```
protected byte[] packageLocationData() {
   String data =
"<9999;999;99.999999999999999;0;99.999;9999999999999;99.9;999.99;999.99;999.99;9;9;999
9999999999;false;9;9;9;9999>";
   try {
      return data.getBytes("ISO-8859-1");  // ISO 8859-1 formatting standard
   } catch (UnsupportedEncodingException ex) {
```

**Appendix C (Continued)**

```
        java.lang.System.err.println("Error converting string to byte array: " + ex);
        return data.getBytes();  //Returns string as standard byte array if ISO 8859-1 unsupported
    }
}
```

The following code segment is used to test the RESTful HTTP transfer time. Unlike the UDP version above, this application requires server-side support in order to receive the acknowledgement of a successful transmission. This server-side support is automatically implemented in RESTful HTTP when a MobileClientToWeb application is developed in NetBeans. Essentially, the developer creates a web service capable of handling incoming transmissions, and in creating a client to interact with this service the NetBeans environment provides the underlying functionality for wirelessly transmitting the data that is expected by the server. In the case of the code below, "client" is the means by which the data is transferred, and *httpPrintString(String)* is a stub (see Section 2.2.2.4 and Section 2.2.2.5) in the phone-side software that invokes the remote printing function through the client.

```
inputString =
"<9999;999;99.999999999999999;0;99.999;9999999999999;99.9;999.99;999.99;999.99;9;9;999
9999999999;false;9;9;9;9999>";
long timeStampA, timeDiff = 0;
java.lang.Boolean response;

timeStampA = java.lang.System.currentTimeMillis();
response = client.httpPrintString(transferString);
timeDiff = (java.lang.System.currentTimeMillis() - timeStampA);
if (response.booleanValue()) {
    return timeDiff;
} else {
    java.lang.System.out.println("Error in RESTfulHTTP transfer function.");
    return -1;
}
```

**Appendix C (Continued)**

The following code is responsible for handling the incoming transmission. It is a simple function that merely prints the string that is sent to the screen.

```
/**
 * Web service operation
 */
@WebMethod(operationName = "httpPrintString")
public Boolean httpPrintString(@WebParam(name = "inputString") String inputString) {
    System.out.println(""+ inputString);
    return true;
}
```

**Appendix D: Code Segment for Evaluating GPS Fix Times**

The following code segment is used within the applications to evaluate the time required by the GPS hardware within the cellphones to obtain position data from the orbiting satellites. The javax.microedition.location library must be imported into the application, as well as any model-specific classes necessary for the phone to access the GPS system. Additional parameters and permissions may need to be set before the GPS hardware may be accessed by third-party user applications.

In the evaluation framework's test on the Sanyo 7050, the com.sprintpcs.util library was also required as an import to the main class, and a specialized Location class was initialized in the appropriate portion of the code. Developers seeking to apply this framework to particular phone models should check with any available developers' manuals for information about specific requirements.

In the sample below, a maximum of 30 meters of estimated uncertainty is permitted for the data to be considered "valid," and the object obtained from the GPS network is used to construct an input string for wireless transfer in order to reveal the information (*e.g.*, latitude, speed, course) contained within. The accuracy value contained within the "QualifiedCoordinates," returned by a call to getHorizontalAccuracy() is used in the Framework as the EAUClear and EAUObscured results when the application is run in an unobstructed and then obstructed environment respectively.

```
double lon = 0, lat = 0;
float alt = 0;
```

**Appendix D (Continued)**

```
try {

  //Obtaining GPS data
  String inputString = "";

  Criteria cr = new Criteria();
  cr.setHorizontalAccuracy(30);
  LocationProvider lp = LocationProvider.getInstance(cr);
  javax.microedition.location.Location l = lp.getLocation(60);
  Coordinates c = l.getQualifiedCoordinates();
  if (c != null) {
     // Use coordinate information
     lat = c.getLatitude();
     lon = c.getLongitude();
     alt = c.getAltitude();
     //Obtaining input string
     inputString = "<" +
           "9999;" + //Dummy session ID
           "999;" + //Dummy segment ID
           lon + ";" +
           lat + ";" +
           alt + ";" +
           l.getTimestamp() + ";" +
           l.getSpeed() + ";" +
           l.getCourse() + ";" +
           l.getQualifiedCoordinates().getHorizontalAccuracy() + ";" +
           l.getQualifiedCoordinates().getVerticalAccuracy() + ";" +
           l.getLocationMethod() + ";" +
           l.isValid() + ";" +
           java.lang.System.currentTimeMillis() + ";" +
           "0" + ";" +
           "-1" + ";" +
           "-1" + ";" + //-1 for unsupported "Cell signal strength" in Sprint config.
           getBatteryLevel() + ";" +
           "9999" + //Dummy testing value
           ">";
  }
  else
   //Provide dummy transfer value as default
     inputString =
"<9999;999;99.999999999999999;0;99.999;9999999999999;99.9;999.99;999.99;999.99;9;9;999
9999999999;false;9;9;9;9999>";

} catch (Exception E) {
    java.lang.System.out.println("Exception in initialize: " + E);
}
```

## About the Author

David Pedro Aguilar was born and raised in the tiny but beautiful Central American country of Belize. At the age of twenty he moved to Florida in order to pursue a course of education that was then impossible to obtain in his homeland. He received his Batchelor's Degree in Computer Science from the University of South Florida in Tampa, Florida in 2000, and entered graduate school that same year.

After obtaining his Masters' Degree, the focus of David 's research shifted away from Neural Networks and toward wireless communications. The effort invested in this area of exploration formed the foundation for the information presented in this dissertation.