Designing Energy and User Efficient Interactions with Mobile Systems

Lu Luo

CMU-ISR-08-102

April 2008

School of Computer Science
Institute for Software Research
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Daniel P. Siewiorek, Chair
Bonnie E. John
Priya Narasimhan
Diana Marculescu, Department of Electrical and Computer Engineering

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by the Defense Advanced Project Agency (DARPA) under Contract No. NBCHD030010, the National Science Foundation (NSF) under Grant Nos. EEEC-540865 and 0205266, the Office of Naval Research (ONR), N00014-03-1-0086, and HP Labs. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies or endorsements, either express or implied, of DARPA, the NSF, ONR, HP Labs, Carnegie Mellon University, the U.S. Government, or any other entity.

Keywords: Mobile computing, energy efficiency, user interaction, user interface design, cognitive modeling, ubiquitous computing

Abstract

Mobile computing has thrived to provide unprecedented user experiences beyond the boundary of desks and wires. The increasing demand for mobility faces two major challenges: reduced form factor and limited energy supply. Although each challenge has been addressed by significant research efforts, the correlation between them is underexplored.

Energy efficiency should be integrated as an important metric of user interaction design in mobile systems. By carefully considering the specific requirements of the user and the context of usage, energy efficiency can be achieved without sacrificing user performance and satisfaction, and interaction design that facilitates user efficiency can also promote energy efficiency.

This dissertation starts with a detailed study of typical workload and screen usage that shows that users seldom use the entire screen on most workloads. Hence, Dark Windows, an energy efficient display design is presented and implemented to optimize display power consumption by adjusting the color and illumination of different screen regions according to the user's workload.

This dissertation next presents AstroRDS, a mobile computing system and network infrastructure that displays documents and information from user's mobile devices on ambient ubiquitous display resources, thus facilitates much better viewing experience of the user with comparatively ease of use. The energy consumption and network usage of AstroRDS is several orders of magnitudes less than VNC remote desktop protocol on viewing and controlling tasks, and with similar initial loading overhead.

A common challenge in these two efforts is the high cost of measuring user experience and energy consumption. This dissertation further presents KLEM, a quantitative methodology based on cognitive modeling techniques that can predict both user performance and system energy consumption from story boards of an interactive task during early stages of design. KLEM can be used as a convenient tool to compare and make early decisions among different design options, as well to resolve potential design issues before investing in actual user testing and iterative development. While KLEM is presented with a focus on handheld devices, the methodology can be applied on any interactive mobile system.

Acknowledgements

More than once I was warned against writing a PhD dissertation whilst working full-time, I am glad that I managed it in three months. This dual-task period has been very demanding indeed, but I found it exciting to get continuously inspired by new ideas from work for my writing, as well as to see how my dissertation research may be further extended and applied in the real world.

Because of this I wish to express my sincere appreciation to my advisor, Prof. Dan Siewiorek for encouraging me to reach out beyond graduate school and collaborate with industrial labs on various projects that I was interested in, meanwhile keeping me focused on the right track of my dissertation work. Dan's insights, methods and knowledge not only made invaluable contributions to this dissertation, but also influenced my development as a research professional. Most importantly, I have been fortunate to work with an advisor like Dan, who was always there for my questions and concerns, who never failed to correct even the tiniest typos in my writings, and who treated all his students with care and the warmth of family.

My thesis committee is quite special in that except for Dan, all other three members are female. I may not have worked directly with all of them, but I regard them as my role models for developing a research career as a woman. I am indebted to Prof. Bonnie John for teaching me cognitive modeling for one semester and shaping the research on KLM for mobile user interfaces. The KLEM method would not be applicable without the ever improving CogTool provided by Bonnie's group. Besides providing prompt and insightful feedback on my research progress, Prof. Diana Marculescu helped validating on the energy characterization methods used in my research, and Prof. Priya Narasimhan's "How to Write a Good (no, Great) PhD Dissertation" presentation served as a great guidance during my dissertation writing.

I was fortunate to have had help and friendship from so many talented people at Carnegie Mellon University. I have built Chapter 4 upon the work in collaboration with Harvey Vrsalovic and Matthew Hornyak. As office mates we pulled many all-nighters together building systems and running experiments, and as good friends it was always fun to exchange crazy ideas and to have technical and profesisonal discussions with them. During the years of my PhD study, Rong Zhang, Joshua Anhalt, and Ravi Mosur helped me a lot building a speech decoder built on CMU Sphinx, which familiarized me with the basics of speech recognition systems. On building cognitive models, Peter Centgraf provided the EventLogger for Palm PDAs. Gus Preva, Alex Eiser, Don Morrison, and Jason Cornwell have helped me setting up CogTool or adding new modeling features for me. My graduate study would not have been possible without the guidance from Profs. Bill Scherlis, David Garlan, and Mary Shaw, who established the software

engineering PhD program, and it was a truly great experience for me to grow with the program as one of the first batch of PhD students. My special thanks to Ms. Laura Forsyth, an amazing lady and former car racer, who is always there when I need help. I would also like to thank Prof. Asim Smailagic, Connie Herold, Helen Higgins, Walter Schearer, Marian D'Amico, Alicia Brown, and all SCS and ICES supporting staff that have made my school life so much easier.

My PhD research also involved collaboration with many industrial researchers. In summer 2002 I worked as an intern at the HP Labs (former Compaq Western Research Lab) with Drs. Bob Mayo and Partha Ranganathan and Chapter 3 is built upon this work. The IBM PhD Fellowship funded me for the school year 2002 to 2003, and in summers 2003 and 2004 I worked as an intern at the IBM T. J. Watson Research Center with Drs. Chandra Narayanaswami, Michael Olsen and Marcel Rosu. I learned a great deal from all my internship mentors and would like to thank them for the valuable research experience at such world-class institutes.

Truly great friends are hard to find, difficult to leave, and impossible to forget. My life at CMU would have been much less enjoyable without the great friendships from Madhavi, Stella, Joy, and many other people. I would also like to thank all my colleagues and friends in Pittsburgh and New York area, you mean a great deal to me.

Finally, I would like to thank my parents for loving me, trusting me, and supporting me all the time, and my wonderful husband Edward for filling my life with with so much love, care, and laughter, you make me a better and happier person everyday.

Lu "Annie" Luo
April, 2008

Contents

List of Figures

Chapter 1

Introduction

Mobile computing technologies have largely extended user's access to the convenience and assistance of a conventional computer beyond the boundary of desks and wires. Today, millions of people have jobs that require them to be on the move – whether they are sales representatives who work in temporary locations, executives who oversee geographically dispersed operations, students who constantly move from class to class and meet with professors and other students, or other workers whose jobs simply require them to be on the go. More so, the upward trend of taking graphical, audio, video, physical, and social information on the road in people's everyday life has shifted the tethered information model people are used to, toward "the mobile lifestyle", in which users can access information any time, anywhere[1].

User's accesses to information on-the-go are enabled by interacting with the diverse services provided by mobile devices. A mobile device is a battery-powered, pocket-sized computing system, typically comprising a small visual display for user output and a miniature keyboard or touch screen for user input. Typical mobile devices include mobile phones, Personal Digital Assistants (PDAs, also known as handheld device, handheld computer, "palmtop", or simply handheld), handheld game consoles, portable media players, personal navigation systems, eBook readers, wearable computers, and so on.

The increasing demands for mobility face two major challenges: the *reduced form factor* and *limited energy supply*. Although neither challenge by itself is new as significant research efforts have been devoted to facilitating user interaction and optimizing the energy consumption of mobile systems, the correlation between them is underexplored. What is the impact of user interaction on energy consumption? How would different user interaction designs affect energy consumption? How to make user interaction design decisions for better energy efficiency? Will energy optimization techniques trade off the user's performance?

Thesis: This dissertation intends to answer the questions above by introducing energy efficiency as an important usability metric for evaluating mobile user interaction design. By care-

[1]Ubiquitous computing (or pervasive computing) is a further form of mobile computing, in which information processing has been thoroughly integrated into everyday objects and activities. Someone "using" ubiquitous computing engages many computational devices and systems simultaneously, in the course of ordinary activities, and may not necessarily even be aware that they are doing so.

fully considering the specific requirements of the user and the context of usage, energy efficiency can be achieved without sacrificing the usability of the interface or the experience of the user; and interaction design that facilitates user efficiency can also promote energy efficiency. This dissertation first presents two research efforts that adapt to the workload and requirements of the user under different circumstances, and achieve optimal results in both energy efficiency and user experience. A common challenge in these two efforts is the high cost of measuring user experience and energy consumption. This dissertation then further investigates a comprehensive methodology that, based on cognitive modeling techniques, can produce quantitative, a priori predictions on both user performance and energy consumption at early stages of interaction design. This methodology not only enables designers to make quick decisions among different designs, but also provides a cost-effective means of evaluation before investing in actual user testing and iterative development.

This chapter begins with an overview of the correlation of mobile user interaction and energy consumption and why it is important to research energy efficiency from the perspective of user interface. Then, Section 1.2 emphasizes the importance of evaluating energy performance during early design. The chapter concludes by providing a roadmap to the dissertation in Section 1.3.

1.1 Energy efficiency as a design metric

Energy efficiency should be considered as a major usability metric in the design of mobile systems[2]. Usability is "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use" (ISO 9241, Part 11), where *effectiveness* is the accuracy and completeness with which specified users can achieve specified goals in particular environments; *efficiency* is the resources expended in relation to the accuracy and completeness of the goals achieved, and *satisfaction* is the comfort and acceptability of the system to its users and other people affected by its use.

To the users of battery-powered mobile systems, it is a paramount concern whether or not they can satisfactorily accomplish their specific tasks with the limitation of battery life. This dissertation explores the efficiency of energy usage in mobile systems at the level of user interface – the degree to which the design of a particular user interface takes into account the energy consumption on specific user tasks on a mobile system. The motivation for taking this approach is that users' view of a computer system is often limited to and based solely on their experience with the user interface – to users the interface often is the system [36]. The user and the computer engage in a communicative dialog whose purpose is to accomplish some task, and all the mechanisms used in this dialog constitute the interface: the physical devices, as well as computer programs for controlling the interaction [26].

The user interface plays an important role of bridging the activities of the user and the computer. Figure 1.1 illustrates the user-computer system structure in the context of mobile computing. On the user side, although the user is the ultimate generator of system activities and

[2]Unless otherwise notified, "device", "system", "computer" are used interchangeably in this dissertation.

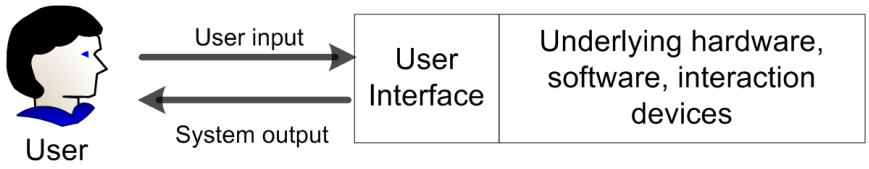


Figure 1.1: The interface is the computer system to the user

consumer of system resources, it is very difficult for a human being to make frequent decisions on energy. For instance, users would welcome a longer battery life, but are often very reluctant to manually make simple adjustments, such as lowering their display backlight, to reduce energy consumption, not even to mention complex adjustments on the time scale of the computer system that is often measured in milliseconds. Computers are successful in doing computational tasks that are impossible for human beings, but not equally successful for helping to increase the usability for simple interactive tasks [92].

On the system side, energy consumption is usually optimized at either the software level by changing algorithms, reducing performance needs, and re-compiling to use lower power instructions; or at the hardware level by providing power saving mechanisms to the software level, and implementing novel circuit structures or devices. However, the tasks users perform on mobile systems involve a significantly large amount of interactive activities, which are difficult to be taken into account at the software or hardware levels, thus make energy optimization approaches at either level less effective. As suggested by the end-to-end arguments in system design [130], higher levels of a system has more information about the overall workload. Since the user interface can be viewed as the highest level of a mobile system and serves as a bridge between the user and the system, it is beneficial to optimize energy at the user interface level in order to achieve better efficiency.

For mobile user interaction designs, note two key aspects for energy efficiency. First, energy efficient user interaction design should not only look at the user's immediate tasks, but also suit the context or situation within which the system is expected to operate, e.g., the domain, and the environment. Next, energy efficiency and other usability aspects can be achieved together by carefully tailored user interaction design. This dissertation presents two studies that exhibit good energy efficiency by leveraging mobile user interaction design with the user's workload as well as the surrounding environment. The first study presents "Dark Windows", an energy-adaptive user interface design that preserves energy on unused portions of the screen, based on the user's workload. The second study presents a remote display system that facilitates utilizing ambient display resources and allowing users to view the documents and information from their mobile devices more easily on ambient displays.

1.2 Energy evaluation during early design

Energy efficiency, though being ultimately important in mobile systems, is contradictorily often considered at very late stages of production, if not after product completion. It is often not until the post-implementation user test or usability analysis stage is the energy consumption presented as a concern. To resolve the concern, necessary optimization may need to reach too far into the system, either software or hardware, to allow timely and economical changes, and the product has to be released with a label "battery killer". In a sense, energy consumption is similar to user interface design, which is separated from the remainder of system design as a standard practice in developing interactive systems, and usability requirements are often postponed to the end of development cycle where they are overtaken by time and budget pressures [79].

Energy efficiency should be more effectively and less costly addressed in early stages of design. The lack of concerns for energy efficiency is due to the unavailability of the actual product for measuring energy usage, especially at early stages of user interaction design, when designers often use sketching and other "low-fidelity techniques" (e.g. creating mock-ups, scissors, glue, and post-it notes) [127] to generate, quickly try out, and compare design ideas. Later they may use user interface prototyping tools or builders to implement the interface, or hand off the design to a programmer. Despite the importance of addressing energy efficiency at the user interface level, doing so is very difficult when many details of architecture, implementation, platform, and runtime task execution are unavailable.

Another challenge to early energy efficiency evaluation is that how a human user actually perform an interactive task on a computer, i.e., the steps the user takes and the cognitive delays between consecutive steps, often remain unknown without the expense of observing and recording the target application on real hardware executed by real users. Moreover, interaction modalities available for mobile devices are quite diverse, which makes it particularly difficult to predict and estimate run-time performance at earlier design stages before the application is actually implemented, deployed, and user-tested.

Several key factors need to be addressed to bridge these gaps: a model of power consumption in the target platform, feedback on what effect various design decisions might have on energy use, a vision of how the application consumes energy and the ability to express that to the system, and the flexibility to convey energy information between different levels of system.

This dissertation presents a quantitative methodology that takes into account the above key factors and evaluates both energy consumption and user performance from interface design mock-ups. This method not only enables cost-effective characterization at an early design stage, but also helps identify possible energy efficiency issues in lower system levels. It gives user interface designers, software developers, and system designers alike an explicit link between their realms of expertise, allowing them to participate more effectively in the early design decisions of an interactive system. This method also eliminates the amount of time and efforts needed to conduct user studies at a later stage.

1.3 Roadmap

Since this dissertation involves many aspects of a mobile system, it is first necessary to set up the context on how each level of the system act on energy efficiency from a user interface point of view in Chapter 2.

Chapter 3 motivates and studies energy-adaptive display sub-systems that match display energy consumption to the functionality required by the workload and the user. Through a detailed characterization of display usage patterns, it is shown that the screen usage of a typical user is primarily associated with content that could be displayed in smaller and simpler displays with significantly lower energy expense. Energy-adaptive designs that use light emitting displays and software optimizations are proposed, and significant, though user-specific, energy benefits and user acceptability are shown.

Chapter 4 presents a mobile computer system and network architecture that provides access to ambient display resources through a user interface tailored for mobile use, on performance constrained devices. This system allows energy-efficient transfers of data and control via a lightweight network protocol, and is built upon an application framework that integrates model, view and control while remaining flexible enough to support diverse data types. Experimental results show improvements in user interface latency and energy-efficiency versus existing methods.

Chapter 5 brings forth using cognitive engineering models to predict mobile user performance, and overviews the most widely adapted cognitive modeling techniques. This chapter then investigates the accuracy of Keystroke-Level Modeling (KLM) predictions for interactive tasks on a mobile device. The models are compared to data obtained from a user study of 10 participants and shown to be able to accurately predict task execution time on mobile user interfaces with less than 8% prediction error.

Chapter 6 presents the Keystroke-Level Energy Model (KLEM), a quantitative analysis methodology that predicts both user performance and system energy consumption of an interactive task at early stages of system design. KLEM extends KLM by integrating system energy consumption into the user model of KLM. A set of benchmarks are designed to obtain the necessary user interaction and energy profiles of the system under study. The KLEM methodology is verified by a user study of 10 participants on two mobile platforms and shows prediction accuracy that is consistent with KLM. KLEM serves as a convenient tool to compare and make early decisions among different design options, and to resolve potential design issues before investing in actual user testing and iterative development. While KLEM is presented with a focus on handheld devices, the methodology can be applied on any mobile interactive system.

Chapter 7 discusses existing work related to this dissertation. Chapter **??** concludes the dissertation by summarizing the key contributions of this research, as well as discussing future research directions generated by this dissertation.

Chapter 2

Energy and Mobile User Interaction

Mobile systems run on the limited energy available in a battery and thus the energy consumed by the system determines the length of the battery life. The electric energy *E*, measured in joules (J), consumed by a mobile device over *T* seconds is equal to $\int^T p(t)$, where $p(t)$ is the instantaneous electrical power measured in watts (W), and 1 joule = 1 watt-second. At any time *t*, the power consumption $p(t)$ is equal to the voltage $v(t)$ multiplied by the current $i(t)$ in direct current circuit (Ohm's Law). Realistically, given a sequence of *n* instantaneous power measurements, each taken δ seconds apart, the energy consumed may be estimated as $E = \sum_{i=1}^{n} p_i(t) \times \delta$, and the average power *P* for such a sequence is approximately $P = \frac{1}{n} \sum_{i=1}^{n} p_i(t)$.

The power consumption of a mobile system is determined by its hardware. Major hardware components of a mobile system include the processor, memory, storage, network interface, display and other interfacing devices. At any time *t*, the instantaneous system power consumption $p(t)$ is determined by all hardware activities occurring at this time. Power efficiency of individual hardware components can be achieved through better material, mechanical, device, circuit, and architecture designs [28, 170], which is beyond the scope of this research.

This dissertation focuses on energy efficiency in respect of mobile user interaction design. The energy consumption of a mobile system is determined by the combined performance of hardware, operating system (OS), application software, as well as the human user. The user interface is the aggregation of means of input that allows the user to manipulate a system, and output that allows the system to produce the effects or responses of the user's manipulation. It can be viewed as the artifact of user interaction that is related to all four determinants of system energy consumption above. An energy efficient user interaction design should take into account not only the interface that define and present interaction behaviors, but also the system components that affect interaction energy.

The following three sections provide a brief overview of the hardware, software, and human elements involved in mobile user interaction, and their impacts on system energy consumption.

2.1 Interfacing hardware

User interaction with mobile systems cannot happen without the support of interfacing hardware components or sub-systems. The most common way for today's mobile user to interact is the users providing discrete input sequences which on the whole can be generalized as various key strokes, and the system responding by presenting information predominantly in a visual form. A visual interface can be non-graphical, i.e., offers only text menus, or requires typed commands. The scope of visual interfaces is broader than graphical user interfaces (GUIs), which focuses on presenting graphical icons, visual indicators or widgets that are often in conjunction with text, labels or text navigation to fully represent the information and actions available to the user.

The major hardware components in ***Visual*** interfaces and their characteristics are briefly reviewed below.

- **Display** presents visual information to the user. Most mobile systems are equipped with a small screen based on various liquid-crystal display (LCD) technologies [85]. Because LCDs require external illumination (back-light or front-light), LCD-based display subsystem often consumes more than half of the system's total power [31, 157]. The organic light-emitting diode (OLED) technology [53] is fast growing and is expected to gradually replace LCDs in mobile devices. A significant benefit of OLED displays over LCDs is that OLEDs do not require a front/back-light to function, thus have the potential of lower energy consumption. An OLED display can also be much thinner than an LCD panel, which also increases mobility. How to optimize the user interface designs so as to reduce display energy consumption will be discussed in this dissertation.
- **Touchscreen** enables displays to have the ability of detecting the location of touches, thus allows the display to be used as an input device. Touchscreens have become commonplace for various types of mobile devices and made them more usable [58, 71, 140]. Resistive touchscreens[1] are the most commonly used, usually pairing with a stylus to manipulate the interface of mobile devices. By comparison, capacitive touchscreens[2] have higher clarity and can handle more than one touch at a time, as is used in the pinch open and two-finger panning gestures on Apple's iPhone. This brings new user interaction experiences but costs more because of their more complex signal processing electronics.
- **Keyboard** is used on mobile devices (often without a touchscreen) to carry out activities including data entry, control and navigation. One technique is to use a miniature version of the traditional desktop keyboard, called "mini-QWERTY" or "thumb" keyboards, which users average an initial 31 words per minute (wpm) and 60 wpm after twenty practices [33]. Devices like standard mobile phones use the ISO keypad layout with 10 number keys and a few extra keys, which reduces the size of the device, but the multi-tap text

[1] Resistive touchscreens use two layers of glass or plastic that can compress and locate the finger or stylus position on a thin metallic, resistive surface. The layers can be damaged by sharp objects.

[2] Capacitive touchscreens use capacitive sensors behind the glass to sense when the electrical field is disturbed. They can detect the finger from as far as 2 mm away. This allows for a more intuitive feel as the finger can glide across the surface.

entry technique used on such keypads can be very laborious and slow. Compared with a conventional keyboard, where people can achieve 60+ wpm, multi-tap keypad can only provide an average rate of 21 wpm for experts [142]. The dictionary-based, predictive text method, T9, has significantly enhanced the basic multi-tapping and allows 40 wpm [142]. The Fastap keypad manages to squeeze a full alphanumeric keyboard with 50 keys in a third of the space of a business card, and uses a technique called "passive chording" to handle ambiguous key pressings [94]. Fastap has the advantages over multi-tap and T9 in that users do not need to work in different ways depending on whether they are entering numbers or words (modeless data entry) and that no training is required [34]. The Twiddler keyboard uses 12 keys and an "active chording" system where the user has to press groups of keys [104].

The much-reduced physical size and the human-centered mobile context provides motivation for exploring novel interaction designs that increases usability. Besides the visual interface discussed above, input and output forms (i.e. modalities) are extended to allow the user and device to communicate more completely and in ways that better fit with mobile demands.

Gestural interfaces exploit human user's expressive movements with fingers, hands or even head. Handwriting recognition is the most widely used gestural interfaces in mobiles systems for text input and editing. The unistroke method [59] and Graffiti on Palm OS or "Block Recognizer" in Windows Mobile platforms are example techniques that recognize simple, distinct handwritten strokes by a pen stylus for inputting alphanumeric characters as well as editing commands such as deleting text or capitalizing letters. Stylus or finger movements can also control an application, as with SmartPad [126] and TouchPlayer [121]. Gestures can be used to lessen the competition for physical space in mobile systems, and accommodates the use-conditions likely with the devices. In tilt-based systems, gestures can also be used for text, cursor and application control [68, 163, 164].

Auditory interfaces leverage human's hearing capability. Auditory cues have long been used to enhance the user experience with an interactive system. Auditory icons were introduced to supplement visual cues [57], and earcons – synthetic non-speech sounds – were designed to convey structural information about an interface [19]. In the context of mobile computing, earcons were shown to help users enter data more effectively [21]. The most expressive sounds for humans are languages, therefore speech recognition and synthesis techniques for mobile UIs have attracted a lot of research interest [54, 73, 133, 162]. Speech recognition based input has become quite common on mobile phones (e.g., voice dialing) and PDAs (e.g. Microsoft Voice Command), but is used for domain-specific command and control only, due to the relatively limited computing capacity of mobile systems. Recognition accuracy is another difficult challenge for real-time continuous speech recognition.

Haptic interfaces involve human's touch and movement sensing abilities. The "vibrate mode" of mobile phones is a simple example of a haptic interface. The TouchEngine interface could give users touch-based feedback by moving the PDA screen, and the users can be given different information by varying the rhythm, intensity and gradient of the vibration [122]. Others proposed to allow the user to manipulate the device by touching, grasping,

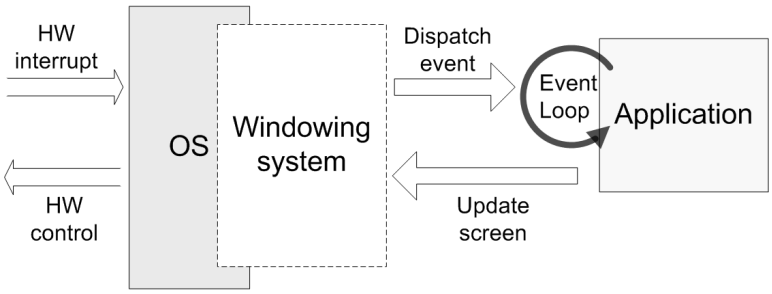


Figure 2.1: The GUI process

twisting, and bending [65, 135].

Multimodal interfaces combine several modalities and enables more natural interactions with mobile devices as in human-to-human interactions. Multi-modality can help resolve ambiguous input and make cumbersome dialog succinct. The QuickSet multimodal system demonstrated potential improvements in speech recognition accuracy by using extra gestural inputs [117].

Novel interaction technologies enrich mobile device's ability of natural user interaction, on the other hand, these techniques can also pose side-effect design problems such as ambiguity and unreliability. For instance, getting robust recognition-based input like speech and gesturing in the real world remains an unsolved problem. From the perspective of enerusability, the extra hardware components and processing overhead on recognizing audio or gesture interaction increase system power consumption $p(t)$, but it may also be expected that these techniques can efficiently reduce task time *T*. Therefore it is necessary to investigate the trade-off between energy efficiency and interaction modality in this dissertation.

2.2 Interfacing software

While interfacing hardware components mostly contribute to system power $p(t)$, the software portion of an interactive user task determines what hardware components are involved and their various activities over time *T*, $T = \sum t_i$ where t_i is the time spent on each activity.

An interactive user task can be viewed as a continuous dialog between the user and the computer. Each dialog is composed of several input-output cycles: first, the user generates some input on a certain interfacing hardware component; next, hardware interrupts are created and handled by the OS, which in turn produces UI events for the application software to handle; the application then determines the output and calls OS services to present the output information on appropriate hardware components. This end-to-end interfacing process happens though the collaboration of different levels of software components involved, which is collectively regarded as the interfacing software during an interactive task. To illustrate this dialog, the software components involved in an input-output cycle in a GUI is depicted in Figure 2.1, since GUI is the most common and familiar mechanism for user interaction.

Different operating systems on mobile devices – Windows Mobile, Palm OS, embedded and mobile Linux, Symbian OS, etc. – provide their own user interface frameworks. Some operating systems integrate the GUI into the kernel, and others separate the GUI as modules. The user interface framework, often called windowing system in GUI because window is one of its primary metaphors, implements graphical primitives such as rendering fonts or drawing a line on the screen, effectively providing an abstraction of the graphics hardware. Although different vendors have created their own windowing systems based on independent code, GUI systems share the same basic elements that define the "window, icon, menu, pointing device" (WIMP) paradigm, and have the same organizational metaphors and interaction idioms. Mobile systems typically use the WIMP elements with different unifying metaphors, due to constraints in space and available input devices.

As is shown in Figure 2.1, interfacing hardware activities occur during user input and system output activities; in between, hardware activities occur, computation and/or communication, to process the input and generate the output. The time of each activity is determined by hardware capabilities on the one hand and software performance on the other. The performance of interfacing software not only means how fast it handles the interaction, but also implies the amount of information that can be conveyed via the output of the system to be processed by the user. However, the computer system only forms one half of the interaction process while the other half is the human user. The total time of an interactive user task also attributes to, after all, the performance of the user.

2.3 Interfacing human

The design and evaluation of interactive computer systems should take into account the total performance of the combined user-computer system [25]. A basic goal of user interface design is to improve the interactions between users and computers by making computers more usable and receptive to the user's needs. A long term goal is to design systems that minimize the barrier between the human's cognitive model of what they want to accomplish and the computer's understanding of the user's task.

For mobile systems, the barrier to designing more natural, efficient user interface is twofold. On the one hand, the amount of device output real estate reduces the amount of information conveyed and services provided to the user. For instance, the most common output technologies for mobile systems are small, low-resolution screens. To deliver the same amount of information, the mobile system needs multiple screen changes as compared with maybe only a portion of a full-sized desktop display. From the discussions on interfacing hardware and software in previous sections, conveying the same information takes longer software time, more hardware activities, and thus higher energy consumption. It would be greatly beneficial to design user interfaces that can effectively utilize the screen real estate and other output resources, and can convey more information with fewer system activities without putting too much cognitive burden on the user.

On the other hand, the input mechanisms of mobile systems limit the user speed for data entry, control and navigation. For systems equipped with a miniature keyboard or keypad, the

user can only use one or two fingers (usually thumbs) to press keys because he needs to hold the device; even if the device is placed on tabletop, there is not enough room on the keyboard or keypad to allow the user to use more fingers as in the much faster, desktop mode, touch typing. For touchscreen based soft keyboard (on screen keyboard) inputs, only one finger or the stylus tip is used. The user spends valuable time looking for keys and moving finger(s) or stylus for each character. Despite the many research efforts on improving text entry speed using mobile miniature keyboard or keypad, most approaches still remain in research communities rather than widely being deployed.

Using handwriting gestures is another option for text entry on mobile devices equipped with a touchscreen, and there are various forms of handwriting recognition mechanisms available on different platforms. The performance of handwriting recognition depends on the writing speed of the user, and the speed and accuracy of the recognition engine. Handwriting input has extra demand on the processor and can lead to more energy consumption than keyboard input.

The final barrier for achieving optimized enerusability is the human user. While the speed of processors become faster and interfaces become fancier, human users still have to read the display, make a decision, and physically move during interaction at the same speed that previous generations did – there is no Moore's Law for humans [141]. Human evolution is a slow process and society-wide human adaptation takes substantial time. Consequently, an increasingly faster processor often spends a great amount of time (over 90% on some tasks according to the observations in [174]) and therefore energy waiting for a constantly slow human user for interactive tasks. Furthermore, humans have a finite and non-increasing capacity that limits the number of concurrent activities they can perform. In the context of a mobile usage paradigm, where there are higher amount of distractions, users tend to try to multiplex more activities, which also reduces their effectiveness [141].

2.4 Summary

This chapter began by describing the key factors that determine energy consumption in mobile systems and pointed out the importance of user interaction design for energy efficiency. Designing energy efficient mobile user interaction requires an understanding of the energy usage of the target system, the software that runs on the system, the energy impact of software design decisions, as well as the energy impact of the human user.

This chapter first analyzed the characteristics of hardware components used in mobile interaction, and the corresponding interface modalities. More detailed, quantitative analysis of the energy impact of different modalities will be discussed later in this dissertation. This chapter then characterized the software processes that are involved in user interaction activities, and the common elements of interfacing software. Finally, this chapter pointed out that the human user is a key factor in mobile system energy consumption. Designing efficient user interaction can not only optimize user performance, but also improve the energy efficiency of mobile systems.

Chapter 3

Energy Adaptive Display

One of the most important components of a user interface is the display subsystem. A display subsystem consists of the electronics associated with the visual representation of the data the system generates, particularly the display panel and the controller. Displays for mobile systems and laptops have been based on different liquid-crystal display (LCD) technologies [85]. The display subsystem often consumes more than half the laptop [157] or handheld [31] system's total energy.

Different users have different workloads and thus display needs. Having a "one-size-fits-all" display targeted at the needs of the most aggressive workload often leads to large energy inefficiencies in the display energy consumptions of other workloads and users. While current approaches to reducing the display power focus on aggressively turning off the entire display when it is not in use or using lower-quality or smaller-sized display panels, emissive display technologies, such as Organic Light Emitting Diode (OLED) displays, allows lower power consumption when a reduced area of the screen is in use [53].

This chapter[1] presents an energy-adaptive display design that takes advantage of the increasing availability of emissive displays. Based on the end user's or application's specific requirements, the display energy consumption can be reduced by controlling individual portions of the display to consume different levels of power. Section 3.1 discusses the user study characterizing display usage in detail. Section 3.2 presents example energy-adaptive display prototypes. Section 3.3 analyzes the energy benefits and user acceptance of the display design and Section 3.5 further explores the design space with energy-adaptive displays.

3.1 User study on screen usage[2]

To understand the screen usage patterns and identify opportunities for power reduction, a detailed study of typical workload and screen usage on laptop and desktop workstations is conducted on a representative user population.

[1] This chapter is based on a joint work [76].

[2] The user study in this section is conducted by HP Labs collaborators prior to the joint work.

3.1.1 Method

The user study is based on usage of the Microsoft Windows environment because of its widespread acceptance and representativeness of the general mobile market. A total of 17 researchers within the lab voluntarily participated in this study. All participants are highly technical users whose typical usage and workloads diverse greatly and cover a cross section of mobile system usage (administrative tasks, code development, personal productivity, entertainment, etc.). Both laptop and desktop PC users are studied to obtain a broader understanding of display usage (many of the laptop users use their machines as their main machine – both as a desktop and a laptop). The systems used by the participants include a variety of screen sizes and display resolutions. Column 2 in Figure 3.1 summarizes the properties of the systems used by the test population.

An application-level logger program is run on the users' machines for times ranging from 1 to 14 days. The logger program is used to collect periodic information about (i) the current window of focus – its size, its location, and its title and (ii) the size of total screen area used (all non-minimized windows). The sampling rate was set to once a second. Screen savers are set to turn on after a reasonable time (1-5 minutes) to isolate only the usage patterns when the user was active. Column 3 in Figure 3.1 summarizes the length of the user traces. The traces range from 9 hours to 346 hours. The variation in the traces represent the differences in how individual users use their machines during their participation in the study. Overall, the samples represent close to 100 days of continuous computer usage time. Column 4 in Figure 3.1 summarizes the length of the "active" user traces, after factoring out the time spent in the screen saver as an indication of the time the user was idle. Traces are still collected during the time it takes for the screen saver to be activated, but given the length of the logs, the effect of this is minor. The sizes of the active user logs range from about 6 hours to 61 hours of computer usage per user.

3.1.2 Results

Average screen usage

Figure 3.1 summarizes the information about the screen usage obtained from the user study. Columns 5 and 6 present the mean and standard deviation, per user, for the screen usage of the window of focus. For this study, the window of focus is defined as the window that accepts keyboard or mouse input. The title bar, the scroll bars, menu bars, and other that are embedded in the window are included in determining the size of the window of focus. Columns 7 and 8 present the mean and standard deviation for the additional screen area used by other non-minimized windows in the system (i.e., the area not hidden under the window of focus).

Studying the average screen usage for the window of focus from Figure 3.1 shows that the test population uses anywhere from 37.4% to 93.7% of the total screen area available to them. An additional 2.3% to 34.1% of the screen is used by other background windows that are not active, yet are not minimized. The last row of Figure 3.1 indicates the average usage across the user population. This average is obtained by computing the arithmetic mean of the averages of the individual users. This ensures that the average is not biased by users with larger log

User	Display (column 2)	Log length (column 3)	Active samples (column 4)	Screen usage for active samples			
				Mean	Std. dev	Mean	Std. dev
				Window of focus		Background windows	
Desktop user population							
1	19” 1024x768	210 hours	33 hours	62.8%	38.5%	10.6%	21.2%
2	21” 1280x1024	346 hours	61 hours	57.2%	22.3%	11.6%	28.5%
3	19” 1280x1024	214 hours	31 hours	46.3%	19.7%	30.4%	19.7%
4	19” 1280x1024	64 hours	43 hours	36.7%	14.5%	34.1%	8.8%
5	19” 1280x1024	253 hours	27 hours	44.5%	22.7%	32.6%	21.1%
6	21” 1280x1024	229 hours	31 hours	55.5%	18.4%	24.7%	17.8%
7	21” 1280x1024	235 hours	30 hours	57.5%	19.2%	20.0%	18.8%
8	17” 1024x768	135 hours	13 hours	85.2%	26.2%	9.7%	24.4%
Laptop user population							
9	13” 1280x1024	42 hours	23 hours	61.8%	21.6%	25.1%	22.3%
10	14” 1024x768	98 hours	54 hours	71.1%	25.4%	22.4%	23.9%
11	14” 1400x1050	57 hours	57 hours	37.4%	20.3%	7.2%	15.1%
12	14” 1024x768	20 hours	13 hours	93.7%	12.3%	2.3%	12.2%
13	15” 1024x768	169 hours	154 hours	43.3%	38.9%	17.5%	24.3%
14	13” 800x600	132 hours	6 hours	71.1%	37.6%	3.0%	15.0%
15	14” 1024x768	9 hours	6 hours	44.1%	21.4%	10.3%	15.3%
16	14” 1400x1050	69 hours	15 hours	54.6%	25.9%	18.5%	17.5%
17	14” 1024x768	10 hours	6 hours	77.3%	36.8%	5.0%	17.0%
Average screen usage – window of focus: 58.8%; background windows: 16.7%							

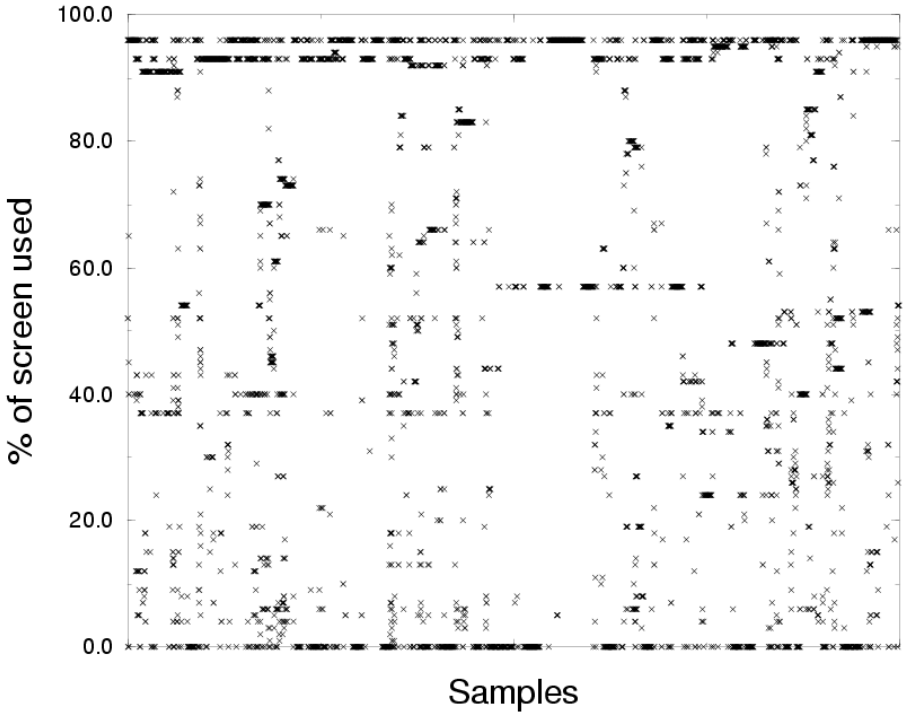
Column 3 summarizes the length of the user traces while column 4 summarizes the length of the active user traces after factoring out the time the user was idle. The window of focus columns summarize the percentage of screen area used by the active window while the background windows columns summarize the percentage of area used by other non-minimized windows not hidden under the active window.

Figure 3.1: Key statistics from user study

lengths. On average, across all the users, typically only about 59% of the entire screen area is used by the window of focus, the primary area of interest to the user. An additional 17% of the screen, on average, is used for background windows that are not minimized. In both these cases, however, the standard deviations are fairly high indicating a wide range in the screen usage values associated with each sample.

Screen usage distribution

To better understand the distribution of the screen usage characteristics, Figure 3.2 plots the variation in the screen usage of the window of focus for one sample user, over the log collection period. Each point represents the average screen area associated with one data sample in the



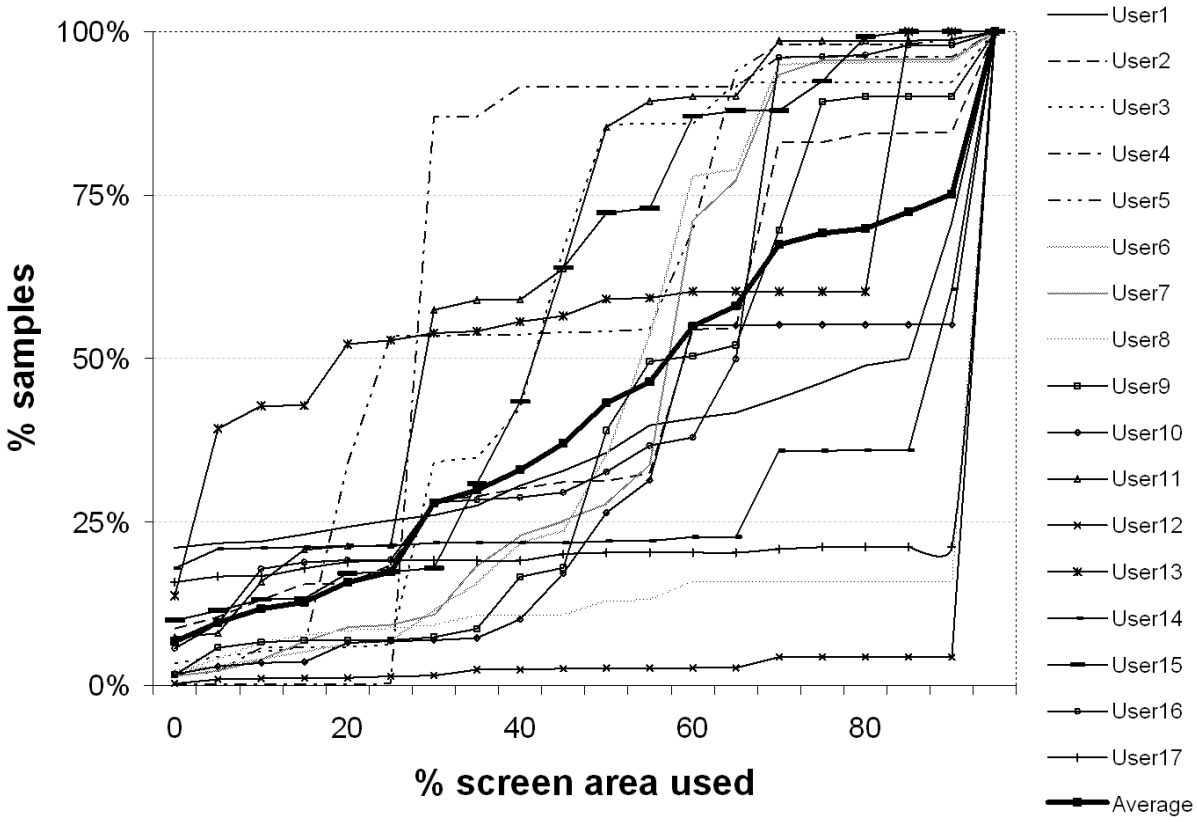
Logged trace of the window of focus from User 1. Each point represents one data sample in the log.

Figure 3.2: Variation in the screen usage for typical user

log. As can be seen from the figure, the percentage of screen usage varies significantly over the time when the data is collected, all the way from near-zero to near-100% usage of the screen. Clustering of points at specific screen usage percentages can be correlated back to the continuous usage of key applications used by the user and their normal (or default) sizes.

Figure 3.3 presents the same data for all the users in a summarized manner. Each line in the graph represents one user from our test population and the thicker solid line represents values averaged over all the users. The X axis represents the percentage of screen area used per sample and is divided into bins of 5 each. The Y axis represents the cumulative number of samples associated with each screen-area-percentage bin. For example, drawing a vertical line from the 50% screen area point to intersect all the lines will give us the cumulative number of samples where each user uses less than 50% of the total available screen area. For User 5, this means that close to 54% of the samples use less than 50% of the screen area.

To summarize the results in the graph, it can be observed that, on average, across all the users, for almost 40% of the time, less than half the entire screen area is used. Some users spend more time in windows less than half the screen area (for example, User 4 spends more than 90% of their time in windows that are typically less than 25% of the total screen area).



Result derived from the test population. The X axis represents the percentage of screen area used per sample divided into bins of 5 each, and the Y axis represents the cumulative number of samples associated with each screen-area-percentage bin.

Figure 3.3: Cumulative distributions of the active screen usages

Screen usage corresponding to application behavior

In order to understand the relationship between the screen usage and the application behavior, the samples from each of the user logs are categorized into four bins – (i) samples where the window of focus usage was between 0 and 25% of the total screen area, (ii) samples where the window of focus usage was between 25% and 50%, (iii) samples where the window of focus usage was between 50% and 75%, and (iv) samples where the window of focus usage was between 75% and 100%. For each bin, the key applications associated with the samples are analyzed. Figure 3.4 summarizes the results. As before, the arithmetic mean of the averages per individual users is computed to avoid distortions due to trace lengths.

Overall, the workloads used by the user population span a range of applications representative of typical system usage. Broadly, they can be categorized into (i) access related – web browsing and e-mail (Internet Explorer, Outlook, mail reader, MSN Messenger), (ii) personal productivity

Active area is 0-25% *(23% of the time for typical user)*
Key applications: 20% taskbar, 15% Program Manager, 5% X-term, 60% miscellaneous windows (message composition, MSN Messenger, Real Player, menu and message windows – properties, connection status, file downloads, alerts and reminders, volume control, printer status, find-and-replace, organizer preferences, file explorer, spell-check, wizards, status messages, file-find, password query windows, confirmation windows)
Active area is 25-50% *(22% of the time for typical user)*
Key applications: 19% X-term, 18% message composition, 6% Internet Explorer, 57% miscellaneous windows (mail related windows, File Explorer, Emacs and Notepad, MSN Messenger chat windows, other status windows)
Active area is 50-75% *(28% of the time for typical user)*
Key applications: 33% Internet Explorer, 24% mail composition and reading, 43% miscellaneous windows (Emacs and Notepad, Image Editor and Image Viewer, messenger chat windows, Frontpage, Framemaker and Ghostview, File Explorer, Powerpoint, Dreamweaver, WinLogger)
Active area is 75-100% *(27% of the time for typical user)*
Key applications: 21% Outlook, 20% Internet Explorer, 7% Excel, 52% miscellaneous windows (Powerpoint, Framemaker, Acrobat Reader, Word [various files], Visual Studio [various files]), Dreamweaver, Imageviewer)

Windows are classified based on their sizes into four bins, and for each of the four bins, the key applications dominating the samples in the bin are summarized.

Figure 3.4: Understanding screen usage by application.

and code development (Word, Emacs, Powerpoint, Excel, Visual studio, Dreamweaver, X-term, Real Player, Image Viewer, Acrobat Reader, Ghostview), and (iii) system related and application control windows (File Explorer, navigation windows, taskbars, menus, status and properties messages, confirmation and password query windows).

Focusing on the windows associated with the various applications, two interesting trends are observed. First, system-related status messages and query windows typically use small window sizes; in fact, these windows constitute a significant fraction of the samples associated with smaller size windows. Additionally, these windows usually display fairly low content that do not need the aggressive characteristics of the display – for example, a comparably lower resolution display with support for a small number of colors would be adequate to obtain an equivalent user experience. Although careful design considerations of the display quality are needed to avoid user performance deficits [41, 60, 64]. Second, personal-productivity applications and development environments and web-browsing and e-mail applications typically use larger portions of the display area. The actual fraction of the screen area used appears to be highly dependent on individual user preferences for window size, fonts, etc. However, even with these large windows, characteristics of the displays such as resolution, brightness, and color are not used to their full capacity.

Screen usage corresponding to user behavior

Focusing on the individual user logs, it is observed that individual user preferences tend to significantly influence the overall screen usage characteristics. For example, User 1 who, on average uses 63% of the display area, has Internet Explorer set to use 96% of the screen area, while User 5 who, on average uses 37% of the display area, has Internet Explorer set to use 67% of the display area. Similarly, User 12, who has the largest screen usage in the study, has a default mail composition window of 95% that dominates the traces. This user-specific sizing of windows appears to be particularly characteristic of web browsing, email, and editor applications. In contrast, for development applications (Visual Studio, and Dreamweaver, Powerpoint), most of the users prefer to have larger windows – possibly because of the multi-window content structure of these applications. Similarly, system-related and application control messages typically use smaller windows irrespective of the user – mainly since the content in these windows is relatively low and in most cases the window sizes are pre-determined by the application. An illustrative examples is the case of User 8 who maximized all windows as a matter of routine ("to be able to read better"). This user still consumes only 85% of the total screen area because of the smaller window sizes associated with system-related and application-control messages. Finally, while the laptop users have a slightly larger screen usage (62% or 8.6") than the desktop users (56% or 10.9") because of the smaller sizes of laptop displays, in general, similar conclusion can be drawn from both laptop and desktop usages.

3.1.3 Summary of user study

Overall, there is a significant mismatch between the properties supported in the display and the actual usage of these attributes by the users in the study. The size of the display used exhibits the greatest mismatch – users use only about 60% of the screen area available. A large fraction of the smaller windows are typically associated with system-related and application-control windows that are independent of user preferences. User preference for smaller window sizes and font sizes can also translate into a greater use of smaller sized windows.

Similarly, there are significant mismatches between the actual screen usage and other attributes of the display such as resolution, brightness, color, refresh rate, etc. In particular, most of the smaller windows include content that could have been equivalently displayed, with no loss in visual quality, on much simpler lower-power displays (lower size, resolution, color, brightness, refresh rates, etc.). Many of the larger windows also do not use all the aggressive characteristics of the display.

These results indicate that energy-adaptive system designs that match display power consumption to the functionality required by the workload/user have significant potential to reduce the energy consumption of the display sub-system.

3.2 Energy-adaptive display sub-systems

This section studies some example energy-adaptive systems to evaluate how the potential benefits identified in the user study can be translated to energy reductions. Section 3.2.1 describes the hardware and software components of these designs. Section 3.2.2 discusses the experimental methodology used in prototyping and studying the user interfaces and the energy consumption for the different designs.

3.2.1 System design

The design of energy-adaptive display sub-system uses emerging display technologies and modified window system software to exploit the mismatches between workload/user requirements and display properties.

Hardware support: OLED displays

To enable energy-adaptive designs based on the user study results, a key requirement is to support different levels of power consumption on individual regions of the display based on the content of output which matches the user's workload. The emerging Organic Light Emitting Diode (OLED) display technology [53] is suitable to support such adaptive designs, because the power consumption of an OLED pixel is determined by its brightness and color. OLED displays are built from small organic molecules that efficiently emit light when stimulated by an electric field. By 2007, More than 100 companies are developing aspects of OLED technology. Samsung, RiT display, Pioneer, LGE, and Philips are among the top suppliers in terms of volume shipments. Kodak, Sanyo, and Sony have shown prototypes from 5.5-inch displays to 13-inch displays at trade shows [8].

In general, OLEDs have better image quality compared to conventional LCDs (better horizontal and vertical viewing angles, higher brightness, and faster response times) and do not need a separate backlight, resulting in lower power. As the technology matures, the biggest challenges are in overcoming yield problems and consequently reducing costs. The latest information on http://www.oled-info.com shows more than 150 product offerings using OLED displays in PDAs, cell phones, MP3 and media players, digital cameras, and other applications [7].

For the energy adaptive design presented in this chapter, it is assumed that a laptop system with a 15" Active Matrix Organic Light Emitting Diode (AMOLED) display is used. The only hardware change needed is replace the conventional LCD panel, backlight, and controller with their OLED equivalents.

Software support: Dark Windows

The software support for energy-adaptive displays can be implemented at one of several levels: the application level, the windowing system level, or the operating system level. This research focuses on modifying the windowing system. Specifically, the Dark Windows optimization uses

the current window of focus as an approximation of the active area that has the user's attention, and to reduce the energy spent on other screen portions. For example, when taking notes using an editor, the user's attention is typically on the screen region of the editor window, i.e. the user's current window of focus. Therefore, changing the brightness and color of other screen portions should bring energy benefits, and should not impact the user experience negatively. To verify this, the next two subsections will discuss the implementation of the Dark Windows energy optimization, and the methodology used for evaluation. Section 3.3 will show the experimental results on energy savings and user acceptance.

3.2.2 Implementation

Several different designs of the Dark Windows optimization are prototyped in order to comprehensively understand its impact on energy and user. Given the difficulties of modifying the Microsoft windowing system that is integrated in the OS, Dark Windows is based on the open source X-based Virtual Network Computing (Xvnc) server [128] for simplicity of implementation. VNC is a display system which allows viewing a computing desktop remotely, regardless of the machine architectures. The part running on the host desktop is called the VNC Server, and the part that views the desktop is called the VNC Viewer. Xvnc is the Unix VNC server, which is based on a standard X server. Applications can display themselves on Xvnc as if it were a normal X display, but they will actually appear on any connected VNC viewers rather than on a physical screen. Xvnc provides a virtual representation of the display hardware – the framebuffer, which brings two advantages to the proposed design: first, it is easy to manipulate individual pixel values on a virtual framebuffer, and second, it is easy to access to the X window server data structures and obtain necessary information such as the current window of focus.

Figure 3.5 shows the software architecture of Dark Windows. On top of the original Xvnc, two modules are added – the *Track* module tracks the current window of focus and other objects that are being displayed, and the *Modify* module changes the values of the pixels in the framebuffer based on the Dark Windows optimization algorithms. To change the brightness and color of non-focus regions, an initial attempt was to simply calculate the new value of all pixels on each display update. Unfortunately this approach greatly increased the computation overhead and caused perceptible slowdown in the rendering speed of the modified interface. Since Xvnc only updates those pixels that have been changed between framebuffer updates (based on changes of the screen), same mechanism should be applied to Dark Windows, and a rectangular update mechanism is used. Before each framebuffer update is sent to the VNC viewer, the Track module obtains from the X-server and records the origin, width, and height information of the current window of focus – the window that keyboard events are directed to. Then the pixels in the regions of update are divided into two groups: Group 1 contains the pixels within the scope of current window of focus, and Group 2 contains those that are outside it. Pixels in Group 1 are sent to the VNC viewer using the original VNC protocol without modification, and pixels in Group 2 are modified according to the energy optimization settings of Dark Windows. Details of the settings will be discussed in Section 3.3. Using this mechanism, the performance overhead

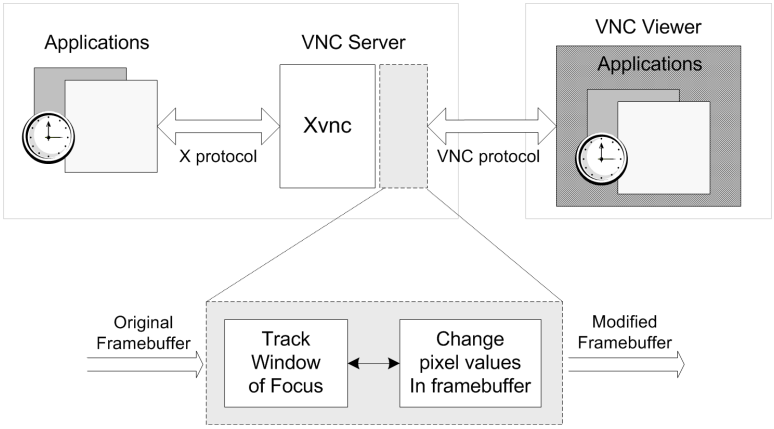


Figure 3.5: Dark Windows software architecture

Design	Modification
HalfDimmed	Areas outside the window of focus are dimmed by 50%.
FullyDimmed	Areas outside the window of focus are fully dimmed (turned off).
GrayScale	Areas outside the window of focus are changed to gray, by setting red, green, and blue values to the average of the three.
GreenScale	Areas outside the window of focus are changed to green (lowest power color for OLEDs). The green value is set to the average of the three colors, and the red and blue values are zeroed. This also dims the screen by 67%.

These four designs change only the background brightness and color, the window of focus is not modified.

Figure 3.6: Four Dark Windows designs

is very small and can be neglected.

Figure 3.6 describes four energy adaptive designs on the brightness or color of the background (regions outside the window of focus). The HalfDimmed and FullyDimmed designs change the brightness of the background[3], and the GrayScale and GreenScale designs change background color. Screenshots of these designs are shown in Figure 3.7. To allow the user to disable the Dark Windows modifications at any time, a keyboard shortcut is added to the Track module.

3.2.3 Evaluation method

The goals of the evaluation are two-fold. One is to understand the intrusiveness of the energy adaptive user interface designs and the complexity and overhead associated with implementing

[3]This study only considers two representative data points – half-dimmed and fully-dimmed. However, the Dark Windows implementation supports parameter based, customized dimming designs that fit individual need and preference.

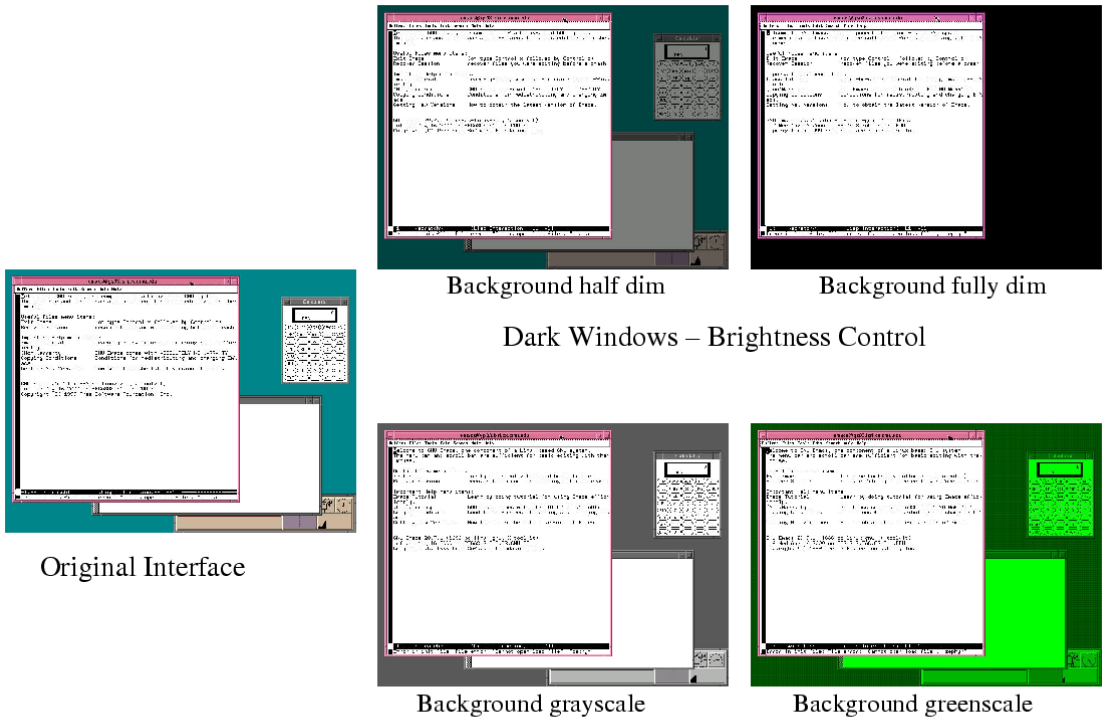


Figure 3.7: Screenshots of Dark Windows designs

the Dark Windows optimizations. The other is to quantify the energy benefits in the context of one particular technology, namely OLED displays, and understand the design tradeoffs between the various optimizations.

Although the Xvnc server for Linux is used for implementation convenience and fast prototyping, the study of energy benefits aims at Microsoft Windows to converge with the user study described in Section 3.1. A synthetic trace is constructed to model the average user behavior observed in the user study, and to replay it in Dark Windows. The trace synthesizes a set of applications that are similar in nature to those used in the user study and the total running time is about 1000 seconds. The trace initializes and terminates the applications with various window sizes and duration of active time to simulate user behaviors. Although the majority of the users in the test population kept the default setting, some chose other backgrounds and window colors. The energy impact and benefit are studied under both situations. Figure 3.8 summarizes the properties of applications used in the trace. Compared with the data in Figure 3.4, the synthesized trace in Figure 3.8 simulates the user study results.

Given the unavailability of 15" OLED displays (aside from rare prototypes) at the time of this study, a software power model is used to characterize the energy consumption of the modified display sub-system. The power model computes the display sub-system power as the sum of

Avg. window size	59%
Avg. background windows' size	17%
0-25% screen usage (taskbar, xterms, miscellaneous)	23%
25-50% screen usage (xterms, editors, mail readers, miscellaneous)	22%
50-75% screen usage (web browsers, mail readers, editors, miscellaneous)	28%
75-100% screen usage (web browsers, mail readers, miscellaneous)	27%
Default screen background color	Teal
Default window color	White
Default foreground font color	Black

The trace models the average behavior exhibited in the user study and was created to match the data from Figure 3.4.

Figure 3.8: Summary of synthetic trace properties

the controller power $P_{controller}$, the driver power P_{driver}, and the display panel power P_{panel}. The controller and driver power are modeled as constant values irrespective of what is displayed on the screen. The panel power represents the pixel array power $P_{pixelarray}$ and is the total of the power consumed by each pixel in the panel array. The power consumed by each pixel is proportional to the brightness of the red, green, and blue components. Let V_{red}, V_{green}, and V_{blue} be the value of each red, green, and blue pixels, correspondingly, and the values of a pixel range from 0 (off) to 1 (fully on). The brightness of each pixel can be adjusted between 0 and 1, and the smaller the value is, the more dimmed the display. For all the pixels in the framebuffer:

$$P_{display} = P_{controller} + P_{driver} + P_{panel} \tag{3.1}$$

$$P_{panel} = P_{pixelarray} = \sum P_{red}V_{red} + \sum P_{green}V_{green} + \sum P_{blue}V_{blue} \tag{3.2}$$

The power values of the model are obtained from the datasheets of existing OLED displays and have been validated by display specialists. For the target 15" OLED display, the $P_{controller}$ is set to 0.5W and P_{driver} to 1W. The maximum of P_{panel} is 8.5W. On full brightness, a green pixel consumes the lowest power ($2.2\mu W$), while a red or blue pixel needs higher power ($4.3\mu W$).

3.3 Experimental results

3.3.1 Power benefits

Each of the Dark Windows designs modifies the pixel values in a different manner, thus resulting in differing energy usage. The power consumption of each design is measured on running the synthetic trace. In comparison, the default interface using the original color scheme on a LCD panel and an OLED panel are also measured. The OLED power model discussed earlier is used to compute the power for the five OLED configurations. The LCD power model is based on the characteristics noted by Choi et al. [31]. For the synthetic trace, the power consumption of the LCD panel hardly varies (less than 1%), and therefore is considered constant.

Figure 3.9 summarizes the energy measurement results. When using the original configuration, the power consumption of the OLED display is 25% lower than the constant 10W power consumption of the LCD display, thanks to the teal background color in the original configuration. With the OLED display, a teal colored pixel (RGB: [0,131,131]) consumes only 30% of the maximum power a pixel consumes when it is in white color (RGB: [255,255,255]). Moreover, the Dark Windows optimizations can provide additional power reductions, for instance, the FullyDimmed optimization provides an additional 20% reduction compared to OLED original, and a total of 43% reduction over the LCD original. The FullyDimmed power reduction comes from dimming the background windows (from white to black) that do not have the focus and the screen background (from teal to black). The HalfDimmed configuration only provides half the power reduction but it allows the non-focus screen portion to remain visible to the user. The GreenScale optimization provides 40% energy reduction over the LCD original, and 15% reduction over the OLED original. The color value of non-focus regions in the GreenScale scheme is computed by taking the average of the R, G, B values of a pixel, clearing the R and B values, and assigning the average to the G value. Recall that the green pixel consume the lowest energy in OLED displays. The combination of using the most energy-efficient color and reducing the brightness by 67% leads to the energy benefits for this configuration. The GrayScale configuration averages the R, G, B values and assigns the average value as the new R, G, and B values, thus makes the pixel gray. However, the measurement results indicate a 1% increase in energy compared to OLED original, although still 28% lower than LCD original. This is because converting the default teal background color (RGB: [0,131,131]) to gray (RGB: [87,87,87]) results in higher value of the R pixels, which consume higher power. Alternatively, energy benefits can be achieved by using the GrayScale configuration when the background color is blue or red.

Figure 3.10 shows the power variation over time of three configurations – LCD, OLED, and FullyDimmed. The *LCD* line represents a non-energy-adaptive LCD display that has relatively constant power consumption, invariant to the size and content of what is displayed on the screen. The *OLED* line shows the benefits that can be obtained from using an energy-adaptive OLED display technology, but without changing the color or brightness of the user interface. In this case, the power benefits mainly come from using the default teal-colored background, which consumes less power than lighter (e.g. white) colors. The *window of focus variation* line at the bottom represents the size of the foreground window (in percentage of the full screen) that

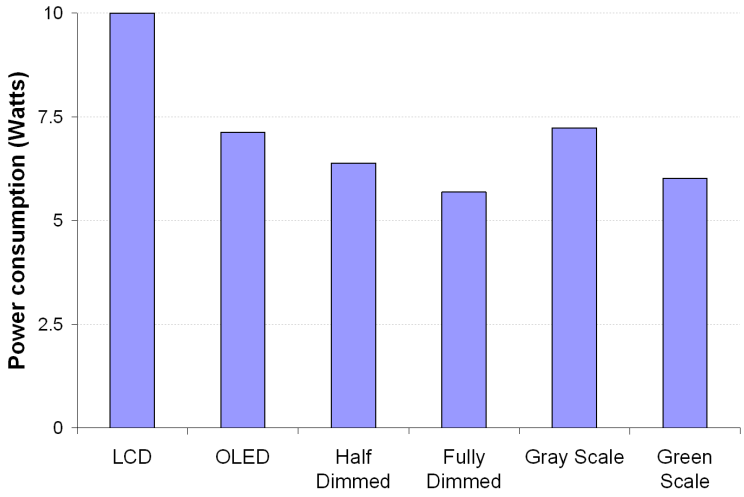


Figure 3.9: Power benefits of four Dark Windows designs

currently has the user's focus in the synthetic trace defined in Section 3.2.3. The smaller the size of the current window of focus (which is white colored as is shown in Figure 3.7), the larger the background (which is teal and consumes less power) size, and the less total display power consumption. The trace synthesizes a set of applications that are similar in nature to those used in the user study and the total running time is about 1000 seconds. The trace initializes and terminates the applications with various window sizes and duration of active time to simulate user behaviors. On top of the power savings from smaller foreground window sizes, the *Fully Dimmed* configuration shows the combined power benefit from using an OLED panel and the Dark Windows optimization that changes the background color to black.

As is evident from the above discussions, the energy benefits from energy-adaptive display designs are highly dependent on the choices of the background color and the window color. The synthetic trace runs with the Windows default scheme to closely represent the majority of users in the test population. To better understand the impact of other schemes, Figure 3.11 shows the energy consumption of various configurations in the extreme cases of pure white and pure black background and window colors. With black background and black windows, the OLED display achieves the highest benefit from energy adaptation – an 80% reduction compared to LCD. The power is mainly consumed by the display controller and driver, and by small-sized non-black window elements such as the title bars. The Dark Windows software optimizations only bring slight improvements over the original configuration. In the other extreme case – the white background and white windows scheme, the OLED original barely provides any improvement over the LCD, and the FullyDimmed configuration reduces the power by 35%. While the all-black and all-white schemes define the upper and lower bounds of the power benefits, the other two schemes show some intermediate points where both the hardware and software optimizations obtain good benefits. These results indicate that energy-adaptive configurations must

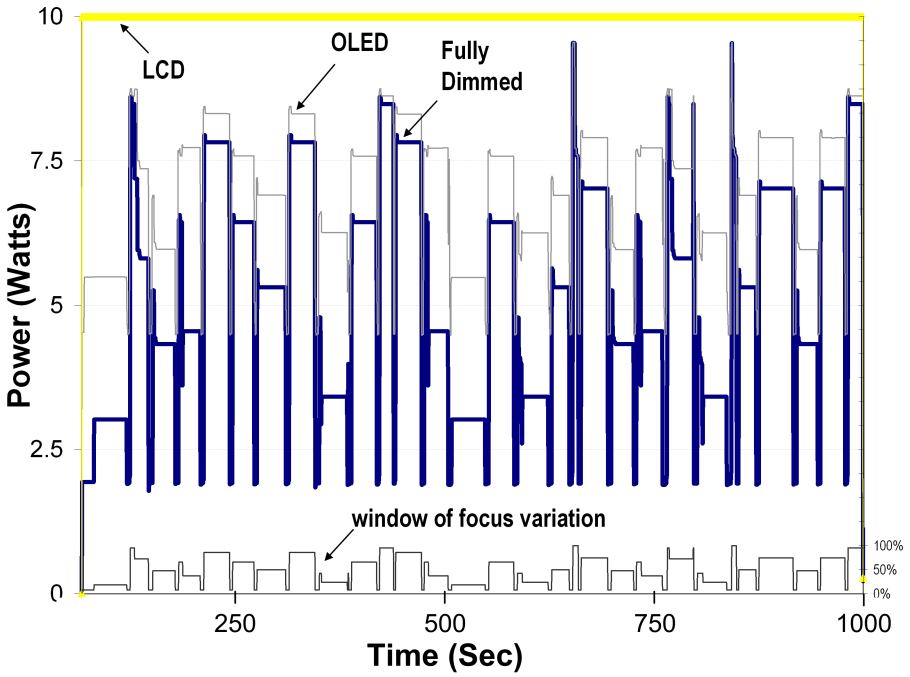


Figure 3.10: Power variation over time for non-adaptive and energy-adaptive displays

be carefully selected for different usage scenarios to obtain optimal energy benefits. It is important for designers to understand typical user behaviors and usage scenarios before applying energy-adaptive display configurations. In some cases, it might be adequate to use the OLED display hardware alone and choose proper color schemes to save power. However, using the Dark Windows software optimizations can provide further benefits in addition to the hardware energy reductions.

3.4 User acceptance evaluation

3.4.1 Method

To determine the impact of the Dark Windows designs on usability, nine researchers are randomly selected from the same group of 17 researchers in the previous user study of Section 3.1.1. The users are asked to freely perform some tasks like opening a text editor, typing some text and then switching to a browser window. The Dark Windows prototype running the synthetic trace is also demonstrated to each user. The users are then informally surveyed on two questions. First, without describing any battery life issues, the users are asked to choose the four Dark Windows designs they like the best. Second, the users are shown the energy benefits from the various Dark Windows designs and then asked to choose their favorite interface again, with the assumption that they are in a situation that required longer battery life, such as during a meeting or an airplane

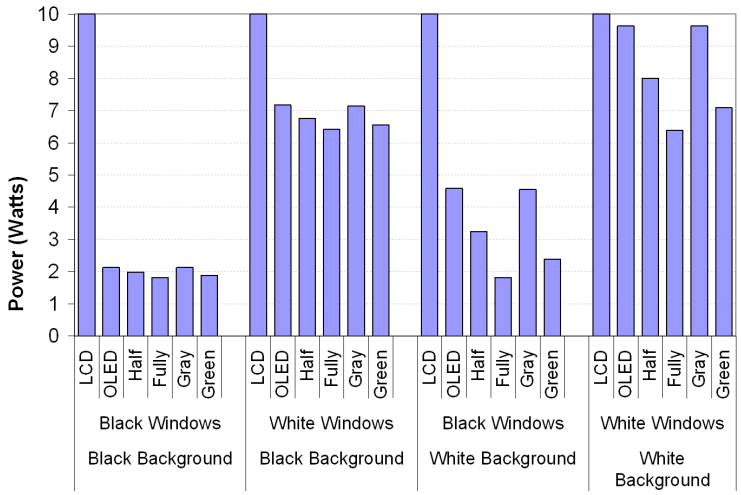


Figure 3.11: Sensitivity of benefits from energy-adaptive designs

trip.

3.4.2 Results

Overall, users hardly notice any difference in the rendering speed between the original interface and the Dark Windows prototype. Most of the users indicate a willingness to use Dark Windows to trade off longer battery life for a different user interface. Four out of nine prefer the GreenScale scheme, three preferred HalfDimmed, and two preferred FullyDimmed. Most of the users express a desire to be able to see the contents of the background, even at the expense of higher energy consumption. Interestingly, even without an awareness of the energy benefits, four users prefer the Dark Windows schemes (GrayScale and HalfDimmed) over the original.

The Dark Windows designs could be improved in several ways (to be discussed in Section **??**). However, the goal of this study is not to perform an exhaustive exploration of the design space for user interface optimizations, but instead to establish that energy-adaptive display sub-systems can provide energy benefits with interfaces that users are likely to find acceptable, particularly in return for longer battery lifetimes.

3.5 Discussion

Overall, the experimental results indicate significant energy benefits from energy-adaptive display designs. The base OLED design achieves 30% reduction in energy compared to a base LCD non-energy-adaptive design – with no change in the user interface. The other Dark Windows optimizations change the user interface in different ways by dimming or changing the color of the background screen area and achieve user-specific, energy benefits. In particular, the choice of the

background and window color can have a key impact on the power reductions. For the default windows background used by users of the user study, the best optimizations, FullyDimmed and GreenScale achieve close to 40% energy benefits over the base non energy-adaptive design. An informal user study indicates reasonable acceptance of these user interfaces, particularly in the context of an awareness of the energy benefits from trading off the interface for longer battery lives.

The configurations discussed in the previous section illustrate some example energy-adaptive display designs. In this section, other possible energy-adaptive options for display design will be discussed.

Other choices on energy adaptive display hardware

Besides OLEDs, there are other display technologies that enable energy-adaptation design, e.g., optoelectronic and light-emissive displays such as Field Emission Displays (FED), conventional Cathode Ray Tube (CRT) displays, and hybrid displays with LCD panel and OLED backlights. For display technologies like LCDs that do not support energy variability, designs can still integrate a multimodal "hierarchy of displays" configuration. For example, a mobile device could have two displays – one higher quality (high resolution, color, high refresh rate, larger size) higher power, and the other lower quality lower power. Theoretically, the idea of Dark Windows can still be applied to this hierarchical configuration with energy benefits.

Exploring other software energy-adaptive optimizations

Other indicators of user activities can be used in addition to the current window of focus. For example, preserve the brightness of the region around the cursor and make the rest of the screen dimmed. Another option is to support user-controlled dimming regions. For example, the user interface can include a "sticky lamp" placed by the user to light up a specific portion of the screen. Much as we do in the physical world, the user can use multiple "sticky lamps" to light up the work area if the current workload requires switching between two or more windows. An alternate implementation can include a "headlight" around the mouse pointer. The user can then point the headlight over regions of interest as needed.

There are yet other application specific dimming interface choices. For example, in a programming environment, there may be a concept of the current procedure and related variables. Portions of the screen related to these can be made bright, for example, all references to a variable and all calls to a procedure. In an email application, perhaps only the current message needs to be illuminated. In a word processor, the line of text being edited needs to be illuminated, the surrounding couple of lines lightly dimmed, and the rest of the document darkly dimmed. Similarly, for applications like Microsoft PowerPoint that use frames within an application, the notion of a frame-of-focus can be defined, similar to the window-of-focus. Moreover, another dimension to these user interface optimizations is to make them time-based. For example, areas of the screen that have recently changed could be bright, fading to a dimer value as time progresses. When inactivity is detected, an email application could dim its screen area until new mail arrives.

The left picture shows the default interface design where the email notification takes the entire display. The design in the middle uses two display panels to save power on the larger display whenever it is feasible to display a small amount of information like the email notification on the smaller display. The right picture shows how display power for low-content messages can be reduced by using simpler visual such as a blinking LED or non-visual cues such as sound.

Figure 3.12: Other energy-adaptive designs

Other user interfaces can be developed by combining the optimizations above. Additionally, other sorts of display mismatches could be exploited. This study has focused on identifying the mismatch between the total area of the display and the area of interest to the user. Other properties of the display, such as resolution and refresh rate should be exploited as well.

Support for output modes beyond displays

The notion of having multiple displays can be taken one step further to match output content to notification mechanisms beyond displays. For example, an email notification that says "You have mail" on the display could be replaced by an LED that blinks on the arrival of email or other similar notification mechanism such as speech output, vibrations, etc. (Figure 3.12). As discussed in Section 3.2, a lot of the smaller windows are typically low content windows which can employ other forms of non-visual communication. This combined with the large design space for alternatives for energy adaptiveness indicate the potential for an interesting future area of research – energy-aware user interfaces.

Accessing user acceptance on handheld devices

The work discussed in this chapter has focused on laptop-level mobile displays. However, as previously discussed, the notion of energy-adaptive display designs can be applied to handheld mobile systems with appropriate selection of optimization schemes, and can potentially provide a positive or even enhanced user experience. While the energy benefits of such designs have been demonstrated in this work, the impact on perceived ease of use, quality, and overall user acceptance on handheld devices should be explored.

Two subsequent user studies were conducted to assess the user acceptance of the energy adaptive display designs. One formally studied 12 handheld users from the Houston area [66], and the other formally studied 12 experienced PDA users from the Boston area [20]. In the user studies, researchers walked participants through scenarios representative of typical day-to-day handheld device use such as e-mail notification, mail checking and reply, note taking, book reading, and checking battery life. Researchers showed participants the default and energy-aware screens in random order and, at the end of the scenario, asked them to fill out a series of quality and acceptance ratings on a nine-point scale. For each interface, participants engaged the prototype to complete the task, offered verbal remarks, and provided ratings based on interface appearance and usability, both before and after learning about the battery-life improvements. A power reduction of 22%-88% over the standard interfaces was evaluated based on detailed power modeling at Kodak that matches the specific chemistry of the OLED display.

In general, participants found the energy-aware user interface designs acceptable. In some cases, they rated the energy-aware designs as highly acceptable and even preferable in specific situations that benefited from improved contrasts and more readable text. Most participants also preferred energy-aware designs that dimmed the background behind pop-up messages. Participants preferred these interfaces because they greatly reduced energy consumption while making it easy to view all necessary text when completing their tasks. Participants rated a flashlight-based interface lower and preferred alternate single-color backgrounds to save energy. It identifies that energy-aware interfaces can actually provide a good combination of energy benefits and greater ease of use by leveraging features that improve usability instead of simply providing a trade-off.

3.6 Summary

This chapter presents the design and prototype of energy adaptive displays. This work gives a detailed characterization of the display screen usage of a representative test user population and indicates novel opportunities of energy reduction based on the functionality required by the workload/user. Built upon the insights obtained from the user study, example energy-adaptive display designs are prototyped. At the hardware level, this design leverages OLED displays of which the energy consumption is related to the brightness and color. At the software level, the Dark Windows power optimization methodologies are presented to enable the windowing environment to change the brightness and color of screen regions that are not of interest to the user. The experimental results show significant, though user-specific, energy reductions with good user acceptance.

Chapter 4

Ambient Display

Chapter 3 has discussed adapting interface design to the user's need and workload to reduce display energy consumption. While task-specific energy benefits can be achieved for laptop-level mobile systems, the Dark Windows optimizations reduce the already small size of screen real estate on mobile systems that are more portable, such as PDAs and mobile phones, which cuts down the amount of information that can be conveyed on the display. Many users carry their mobile devices with gigabytes of personally interesting information. However, the small size of the devices are not optimized for comfortably reading long documents, viewing large photographs, or watching high resolution movies. Users often need to transfer the files using a memory card or synchronize their devices with a desktop computer using synchronization software such as ActiveSync and HotSync, and then view the documents/photos/movies on the desktop display. However, synchronizing information from a mobile device to a desktop PC is slow, sometimes undesirable especially when the PC belongs to another user, and sometimes impossible when the user is on the move or a necessary connecting equipment (memory card or cable) is not available. For example, for users who merely want to preview a video or skim a document sent through email, they must find a secure desktop system so as to not compromise their email account information. They have to make sure that the desktop has the means to exchange files with the device. They also have to face problems with varying versions and capabilities of email readers.

This chapter[1] presents the *Astro* Remote Display System (AstroRDS), a mobile computer system and network architecture that facilitates utilizing of ambient ubiquitous computing resources and allowing users to more easily view the documents and information on their mobile devices. Many mobile devices are equipped with multiple wireless networking media including cellular data, Bluetooth and Wi-Fi. Many environments contain embedded computers and data projectors, including offices, meeting rooms, and classrooms [9]. Just as public Wi-Fi "hotspots" and rooms on corporate and school campuses equipped with projectors are now commonplace, various environments in the near future will have ambient computing resources available to roaming mobile users. The user needs an easy-to-use system to manage the discovery and utilization

[1]This chapter is based on a joint work [161].

of ambient resources, as well as to manage the data to be transmitted and manipulated across various devices.

The system architecture of AstroRDS is described in Section 4.1. The network performance and power consumption characteristics of AstroRDS is analyzed in Section 4.2. The experimental results show the performance of AstroRDS is favorable on a series of common mobile document display tasks in terms of both network bandwidth usage and power consumption. In Section 4.3, a discussion of future extensions to the system that can increase its flexibility and utility is given.

4.1 System architecture

4.1.1 Remote display system

There have been various solutions to display mobile content on ambient resources. Chapter 3 has introduced the Visual Network Computing (VNC) protocol [128] and implemented energy adaptive display designs based on it. VNC focuses on full interaction and control on one computer from another computer or mobile device. Optimizations on image encoding in the VNC protocol are presented in [96]. Optimizations on display command selection and queuing using display-driver level access are described in [15]. Microsoft's Remote Desktop Protocol (RDP) allows a "thin client" user to connect to a computer running Microsoft Terminal Services through a network connection [4]. A functional simulation of wrist-worn projector and a set of interaction techniques are described in [18].

The Pebbles project [110] studies how computing functions and the related user interface can be spread across all computing and input/output devices available to the user, and suggests the notion of "multimachine user interfaces" (MMUIs). Their focus is how handhelds and PC work together when both are available. For individual users, Pebbles studies the research issues of using multiple computers simultaneously to control an application; sharing information among the computers; and using handhelds as personal universal controllers. For group work, Pebbles studies the research issues of private displays versus shared displays and interaction techniques for multiple users.

There are existing systems that create a virtual device remotely, then emulate input and output on this virtual replica based on input and output received from the actual, remote device. This makes the assumption that the input and output modalities of the remote and local device are approximately equal. In the case of mobile devices like wearable computers with significant input/output limitations and an ambient display with far greater output capability, this assumption is severely violated.

AstroRDS enables a controlling mobile device and an ambient display device to work together to manipulate data between them, with the specific constraints of the respective devices in mind. Therefore the system design focuses on operating within the power and user interface constraints and architecture limitations of mobile computing devices. The AstroRDS is composed of three components: a network protocol for discovering and interacting with ambient displays,

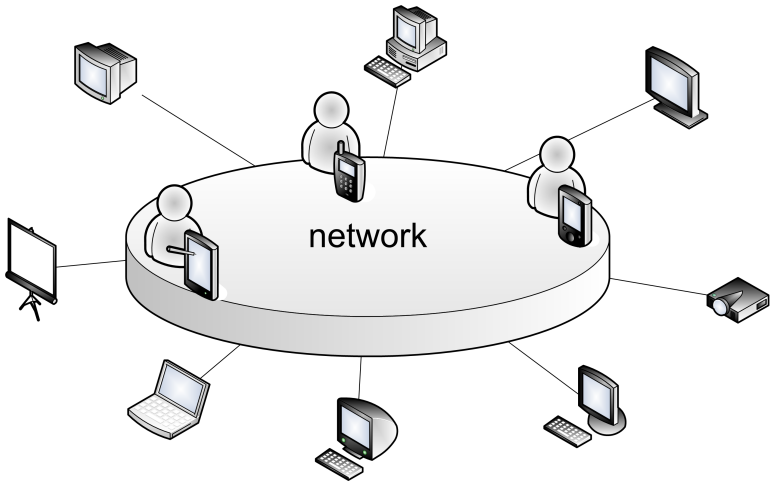


Figure 4.1: An example mobile environment with ambient display and computing resources

a lightweight user interface for mobile devices, and an ambient display system that is capable of viewing files of various data types transmitted from mobile devices. The discovery protocol allows the mobile devices to locate the appropriate ambient display resources within the environment where the user is located. After selecting a resource, the user can view the file displayed on the ambient display using a paired software architecture consisting of mobile control and ambient display components. The mobile control program is developed using an efficient mobile user interface toolkit. The ambient display component is developed using a novel ambient display framework.

4.1.2 Discovery protocol

AstroRDS operates in an environment scattered with display and computing resources of various capabilities, as is illustrated in Figure 4.1. While some may be connected to a backbone network, others may only have a local radio capable of point-to-point connections with peer devices such as a user's phone or PDA. While roaming in this environment, the system must: (1) detect the display and computing resources in the immediate environment and determine the relative physical location with respect to the user; (2) query resources to determine their processing and graphical output capabilities; and (3) form and break connections quickly between the user's mobile device and the resource. In addition to these three provisions, the discovery protocol must be minimally disruptive to the network environment whenever possible. Also, as it will be running continuously on the mobile device, even when the user is idle, it must be power-aware in order to maximize the device's limited battery capacity.

A lightweight discovery protocol for AstroRDS is designed that uses a simple mechanism to satisfy the above requirements. The protocol works over IP, with the ambient resources (displays, processors, etc.) broadcasting UDP packets at periodic intervals. Each packet can be variably aimed in network scope: some packets can be sent on the local subnet only, while others can

be broadcast to the next subnet up and so on (as long as the intermediate bridges, gateways, and routers are configured to allow such behavior). This offers high-level, coarse control on the visibility of individual resources. This is desirable in cases where a particularly useful resource should be advertised to as many users in the immediate and extended area (e.g. campus wide, citywide) as possible.

To avoid network congestion, broadcast messages cannot convey all the information necessary for users to determine if a particular resource has the capabilities to carry out a desired set of operations. However, the messages are sufficient to narrow down the set of candidate resources to those most likely to support the desired operations. Once the user selects a resource to examine, the second level of the protocol allows the user to determine if it meets the precise requirements by making a specific query to the resource. The information returned by the query reflects not only the resource's overall capability, but also its current service capacity. For example, though a display is able to handle large format bitmaps, it may currently be in use and thus momentarily incapable of servicing such requests. The query and response messages, though designed to be efficient, are not critically constrained in size as they are infrequently transmitted relative to the periodic and constant transmission of the broadcast resource announcements.

After a user selects a candidate screen and confirms its availability from the mobile device, a connection is initiated. Though the connections are performed via TCP, giving the notion of a reliable-delivery circuit, the nature of the system and the underlying wireless connection medium require the system to accommodate frequent and unpredictable disconnects, both wanted (the user decides to simply walk away) and unwanted (network error causes the connection to be broken). AstroRDS places all connection management functions in responsibility of the mobile device, as the user is the only one who can completely determine the nature of the disconnect. The ambient resource remains robust and saves connection state in case of an "unexpected" disconnect (one that comes without an explicit disconnect message), but only as long as a new connection from a different user is not initiated. In the future, the ambient resource may save state indefinitely, to be resumed in the future, or transfer state to another compatible ambient resource, to resume the session there.

4.1.3 Software architecture

Since most of the interaction that users perform on their mobile devices focuses on the remote display and manipulation of data, a data-centric software architecture is developed for AstroRDS. This simplifies the integration of other mobile applications (such as mail clients) with the system, as they need only pass the document and its data type to the system. The software architecture has two components: the processing and output component that runs on the ambient resource, and the user interface and data storage component that runs on the mobile device. Each application has specific requirements for output and the types of user controls needed. The ambient display is used as a dedicated display for the user's document, and the mobile device is tasked with displaying the user interface.

This architecture provides optimal separation of function based on the capabilities of each

device. Code necessary to create a responsive user interface is kept local to the mobile device, and code for displaying and manipulating the file is assigned to the remote display as it has the necessary CPU and power resources.

In order to minimize network traffic to preserve bandwidth and power resources, the protocol is implemented in binary instead of XML or other more verbose protocols. Each function in the user interface transmits a control message. Messages are typically under 20 bytes in length. The UI can also send query requests to the display in order to provide the user with status updates about the interface. For example, an ambient display showing a document can provide the mobile device with thumbnails of each page.

Files are streamed over the wireless network without being altered. Because most modern file types (documents, images, and video) are already compressed, altering the file by compressing it is unnecessary. Furthermore, sending the file in its native format allows the ambient display to manipulate the data by the mobile system transmitting only control commands over the network, rather than retransmitting portions of the file (either as raw display data or otherwise).

4.1.4 Lightweight user interface

The user interface of the mobile device changes its functionality and control types based on the data type being manipulated. For example, a text document viewer needs page up/down, text search, and zoom controls. A picture editor requires finer-grained pan-and-scroll, as well as region selection and cut-and-paste controls.

This system needs a lightweight user interface framework for creating interfaces that adapt to the data type being displayed. It must meet the following requirements:

(1) Capacity to support a full range of user interaction through a collection of easy-to-use "widgets," such as buttons, list menus, sliders, knobs, and indicators that can be individually used or grouped with other widgets.

(2) The user input mechanism is tailored to handle inputs from a wearable or handheld mobile device.

(3) Adding and removing widgets from the interface should be possible at runtime and should be fast. The entire composition of the user interface (which widgets are placed where) must be able to be changed to a different layout instantaneously.

(4) The memory footprint of the entire user interface mechanism must be as small as possible, with an upper limit of 10 megabytes.

Of the above requirements, (3) and (4) are considered to be of primary importance. The user interface framework is composed of a variety of application component modules called AppModules. Each AppModule is associated with an ambient display component and data type. For instance, a text display is controlled with the TextView AppModule, an image display is controlled with the ImageView AppModule, and so on. An AppModule consists of a description of which control widgets are used and where they are located in relation to one another on the mobile device display. It also contains rules of what actions are to be taken corresponding to each control being selected or activated by the user.

AppModules are self-contained and hold the code necessary for redrawing component widget controls and processing external events. Each AppModule is given a portion of the total screen space to "own" when it is instantiated. When told to exit, it cleans up and repaints the screen space automatically, and releases any memory heap resources. This provides a flexible and convenient mechanism for managing different applications within the user interface framework. Current mobile devices are always memory limited relative to PCs, and the number of application components running on the mobile device can quickly consume all of the available memory if being loaded simultaneously (the minimum memory footprint of an AppModule is currently around 100KB). However, since the user will load and view one document at a time, only one AppModule needs to be loaded and active, although AstroRDS can understand many data types (each with its associated AppModule). AppModules can be quickly loaded and unloaded at runtime, so the memory footprint can be efficiently managed. There can be provisions for multiple AppModules, up to the limit of memory and screen space.

4.1.5 Ambient display and display protocol

The ambient display, like the mobile user interface, operates in a modular, data-centric manner. It consists of two layers: a network listener and some filetype-specific DispModules. The network listener accepts requests to display files, loads the appropriate DispModule, then routes the file to it. Each DispModule reads the file from the network layer and responds to commands sent by the mobile user interface over the network. For example, the video player DispModule is capable of decoding and displaying the video and responds to commands to rewind or pause the video. Each DispModule can also process the file it has received. For example, the image display module can rotate the image using the CPU power of the ambient display (which is faster than the CPU on the mobile device). DispModules take advantage of existing file display libraries. This allows new file types to rapidly be added to the ambient display system. DispModules can run statelessly; that is, since only one file is displayed at a time, they are able to withstand disconnection and reconnection without losing the ability to control the display.

4.1.6 Limitation

One of the biggest limitations of the prototype system is that for each new data type, a new AppModule must be specifically written and compiled for the mobile device architecture (in our case, WindowsCE targeted for the ARM platform). An ambient display component must also be written (Windows, x86 platform). If a new data type is introduced after the binaries are deployed on the devices, there is no provision to add the ability to handle it. One way to solve the problem is to introduce a virtual machine (VM) mechanism to run interpreted code. Mobile devices would carry the data as well as a VM-coded program that contains instructions on how to render the data to the screen, and the hooks that the control component running on the mobile device can use to manipulate the data. The mobile device could also run a version of the VM, allowing control components to be changed or added at runtime. Different hardware, both in the domain of the ambient resources as well as mobile devices, require VM runtimes to be written natively

for that hardware, while on the other side keeping compliant with the VM exported interfaces used by the display and control component programs.

4.2 Experimental results

Experiments are performed to test the power consumption and network bandwidth performance of AstroRDS in comparison with a system that uses the VNC remote desktop protocol. The experiments consist of a series of typical file display operations using common file types: Windows Media Player, JPEG, Microsoft Word, Excel, PowerPoint and PDF. The experiments measure the time, bandwidth, and power required to display the files and perform typical viewing operations using the applications (such as zooming or scrolling through pages).

For comparison, the VNC system used in the experiments is called PocketVNC. It allows a desktop computer running Windows to access and control a PocketPC 2003 device. It works by using a TCP connection initiated from a client on the desktop computer, connecting to a server process on the PDA. The server process encodes the output that would ordinarily be sent to the PDA screen and reroutes it over the TCP connection for the desktop client to display. Also, the keyboard and mouse input on the desktop is encoded and sent to the PDA, where the input data is used to emulate stylus and/or hardware button presses.

Both systems use identical hardware. The PDA is a Dell Axim X5, which has 32MB of RAM. It is also equipped with a 512MB SD Card for storing the files and a Symbol Networker24 802.11b card. The ambient display was a PC with a 1.7GHz Pentium M processor and 1GB of RAM.

4.2.1 Network usage analysis

One of the key differences in the way that AstroRDS works versus VNC-like systems is that Astro sends a large amount of data to “preload” the file so that later control messages can be short, much like a cache. Once the file has been transmitted, only control information is sent over the network, rather than control and display information. This results in less data sent per control transaction, such as when a document scroll or zoom command is performed and the display needs to be updated. VNC systems do not need to “preload” the entire data file as our system does. However, VNC must continuously encode and transmit any updates to the screen, such as when a document is scrolled or zoomed, even though the original source data of the document has not changed.

To analyze the potential tradeoffs between the two approaches with respect to network bandwidth usage, a series of operations are performed on both AstroRDS and the PocketVNC system, and the amount of data sent from the mobile device to the display device measured . The operations are chosen that represent common actions one might take when viewing various types of documents: a video clip, a JPEG picture, a PDF document, and a PowerPoint presentation. All results are plotted as amount of data sent over the network (in bytes), on a logarithmic scale.

For the video file test, a 231 second, 320x240 WMV9-encoded digital video clip is used. In the VNC test case, a 64-color video display setting is chosen with "hextile" encoding. The AstroRDS test uses the full-color display mode. Figure 4.2(a) shows the result of the test. The columns on the left represent the video clip played back from start to finish. The network usage is approximately comparable. However, the VNC system is sending the framebuffer over the network in 64 colors (5-bit color), and sending 24-bit color (in which the source file is encoded) framebuffers would cause significant overhead in VNC.

The right columns show a seek operation into the middle of the file (after it has completely loaded) followed by 15 seconds of playback from the location. The AstroRDS case has significantly less usage, since it sends only the control data and the video data is already on the display device. VNC must resend that portion of video. There is already a net savings for AstroRDS on the first seek operation, and this compounds with each operation. This is favorable behavior since it is reasonable to assume that a user will perform many operations on the file once it is loaded, giving AstroRDS the opportunity to offset the costs of the initial transfer.

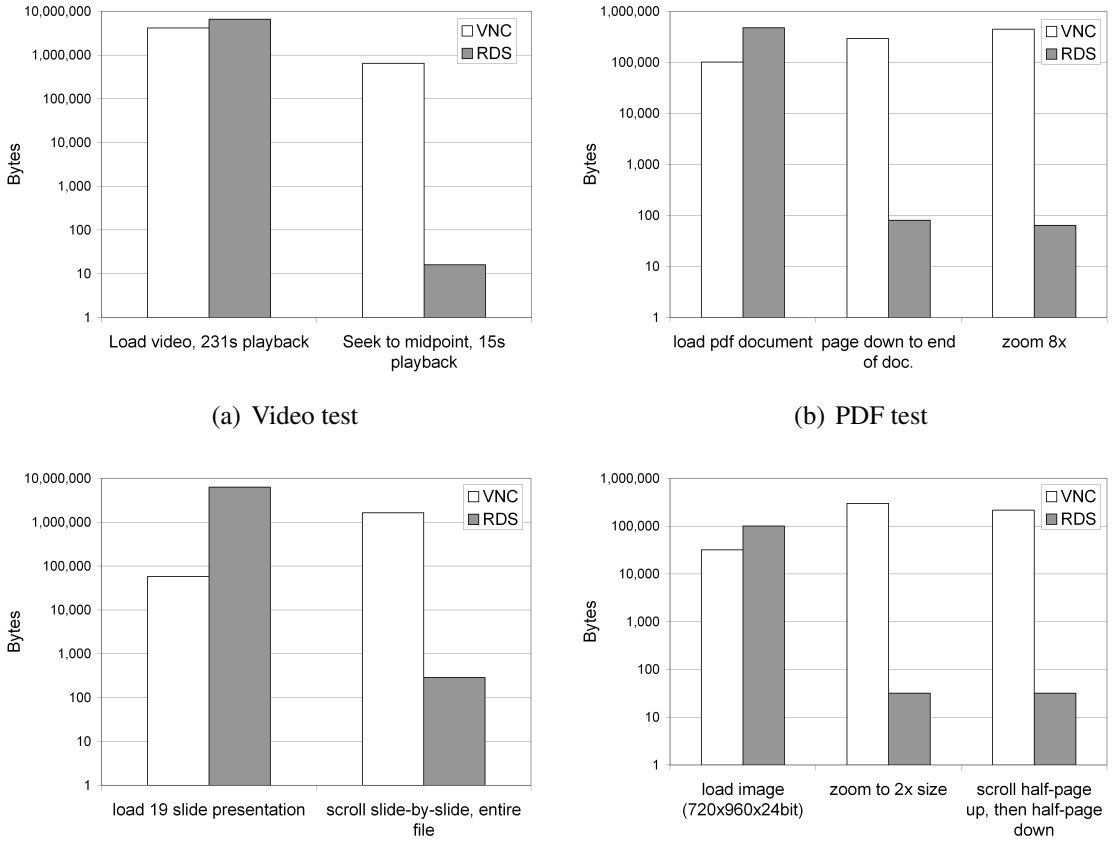
Figure 4.2(b) shows the results of the Portable Document Format (PDF) file test case. This consisted of loading the document and waiting for it to be fully rendered to the display, for VNC and RDS. After the image was loaded, a "page down" operation was repeated until the end of the file was reached. The screen was fully re-rendered before the next page down command was issued. The RDS case again sends only the control information and leverages the fact that the display device already contains all of the document data. VNC must resend actual graphics data to re-render the display. This trend is continued with the zoom operation test.

The PowerPoint test, shown in Figure 4.2(c), consisted of loading a typical 19-slide presentation, then scrolling through each slide. The next slide command was delayed until the current slide was fully displayed. Figure 4.2(d) shows the Image test, in which a fairly large (720x960), 24 bit color image was first loaded, then zoomed to 2x size, and scrolled half a page up and then half a page down.

VNC sends substantially less data while loading the file. However, VNC again sent much more data while browsing the document. This somewhat understates the difference, as PocketVNC transmits a 320x240 image, whereas AstroRDS displays a full 1024x768 image. These results show that the initial "preload" is a good tradeoff for efficient performance while browsing the document. This works much the same way as caching, in that savings on future operations offset an initial "expensive" operation. For example, a user who browses through a longer document and frequently scrolls through the document will benefit from this tradeoff every time he changes pages. It is expected that most users will perform a greater number of control operations than tested, and as such, AstroRDS should perform even better in practice.

4.2.2 Power consumption analysis

The power analysis is focused on the two most representative file types: Windows Media Video and PDF. Figure 4.3(a) shows the moving average of Watts of power consumed by playing back the sample video used in the network bandwidth analysis. VNC consumes significantly more



(a) Video test

(b) PDF test

(c) PPT test

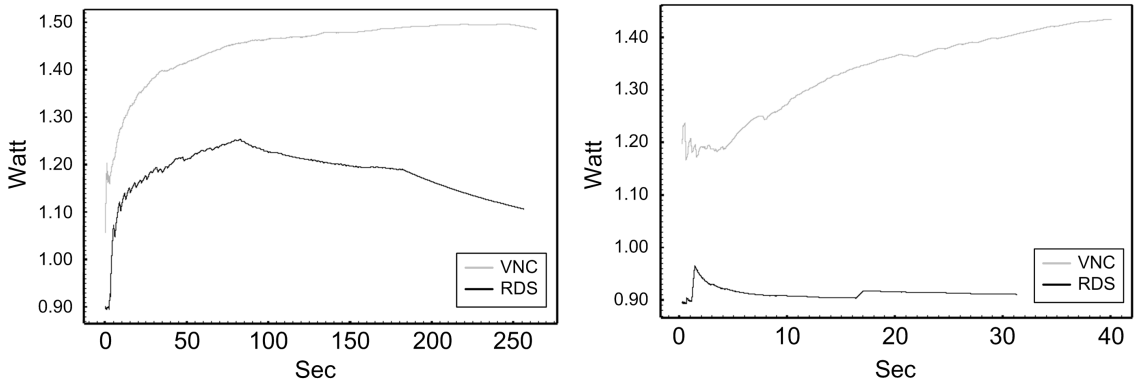
(d) Image test

Figure 4.2: Network impact analysis

power than RDS, particularly after RDS finishes transmitting the file (which occurs at around 75s). VNC's power consumption is high because the system is tasked with reading the file from the memory card, decoding the video file, writing it to the framebuffer, encoding the framebuffer for transmission, and sending the screen contents using the network adapter. The power consumption of RDS is relatively low during transmission because the system is merely reading the file from the memory card and sending it using the network adapter. The CPU remains relatively idle, allowing the operating system and CPU's to conserve energy. After file transmission completes, power consumption on the PDA using RDS returns to idle levels while the video plays on the ambient display.

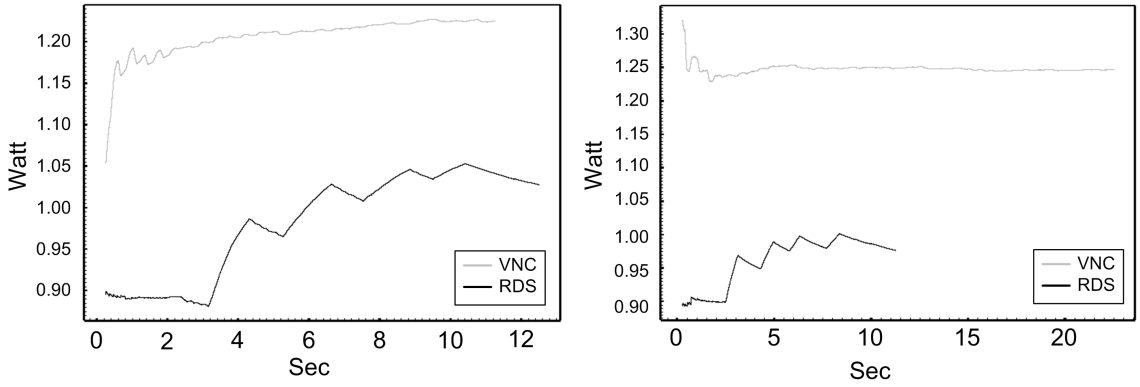
Figure 4.3(b) shows the moving average of Watts of power consumed by the PDA watching fifteen seconds of video then moving to another part of the video and watching for an additional fifteen seconds. The file has already been cached by the ambient display.

RDS remains idle for most of this task, whereas VNC is forced to keep the CPU and radio in operation for the entire duration. Aside from power used to transmit the play and move messages,



(a) Video playback (b) Video navigation

Figure 4.3: Power analysis of video task



(a) PDF loading (b) PDF zooming

Figure 4.4: Power analysis of PDF task

RDS allows the PDA to remain idle. VNC must perform the same set of tasks it performed while playing back the full video, preventing it from realizing any power savings. To illustrate the power consumption performance for viewing a document, a PDF file is used. Figure 4.4(a) shows the moving average of Watts of power consumed on the PDA in opening the PDF file on the ambient display.

VNC consumes more power than RDS because it is using the CPU to decode and display the PDF file as well as encode VNC packets for transmission. RDS has four power peaks, each corresponding with the transmission of a batch of ten file payload packets. Figure 4.4(b) shows the moving average of Watts of power consumed by the PDA while zooming in four times on the document.

AstroRDS is able to utilize the faster CPU of the ambient display to perform the zoom operation, allowing it to complete and return to idle in half the time of VNC. The system consumes

significantly less power, with peaks coming at the transmission of each command. VNC consumes the same level of power throughout the operation as it uses the CPU to render the zoomed document and encode the display packets for continuous network transmission.

4.3 Summary

This chapter has reported the AstroRDS system that allows remote display and control to largely improve the user experience on viewing documents from the mobile device. A prototype system is implemented and the energy and network impacts are studied. Experimental results show that AstroRDS has orders of magnitudes benefit on energy and bandwidth for viewing and controlling tasks, with similar initial loading overhead when compared with the VNC remote desktop system.

Chapter 5

Predicting Mobile User Performance

So far this dissertation has discussed two mobile user interface designs that optimize system energy efficiency without negative impact on, even improve, user performance and interface usability, when carefully tailored with user activities and usage contexts. Both approaches are challenged by a common problem: to verify the benefits of the design, the time and efforts expended on user study often exceed that on design and implementation, and the diversity of mobile user interfaces exacerbate this issue.

The diversity of mobile user interfaces is now greater than ever. As discussed in Chapter 1, mobile systems vary tremendously in purposes: there are consumer products and business products; in form factors: they range from sub-laptop tablets to wrist wore watches; in functionality: there are phones, PDAs, media players, game consoles, wearable computers and other appliances; in platforms: there are 30-40 different operating systems available [132] and the major ones among them are Windows Mobile, Palm OS, Linux, Symbian, and Android; in interaction modalities: there are keypads and keyboards, stylus and fingers, touchscreens and head-mounted displays, writing and speech; and most of all, in applications, services, and contexts. It is utterly important to get the mobile interface design right, it is even more so to evaluate and compare different design options with as little cost as possible.

This dissertation presents a cognitive modeling approach that focuses on producing quantitative a priori predictions for interacting mobile user systems that have not yet been built. Cognitive modeling methodologies are grounded in the extensive theoretical and empirical work of HCI researchers and can be used without further empirical validation to make predictions. This chapter first shows how cognitive modeling can be used to effectively predict user performance on mobile systems. The next chapter will describe in detail the methodology of using cognitive modeling methodologies to predict the energy consumption of mobile systems.

Section 5.1 briefly introduces the psychology science base of modeling user performance and the advantages and limitations of cognitive engineering models. Section 5.2 investigates the applicability and prediction accuracy of KLM on pen-based, touchscreen mobile interfaces. Lastly, Section 5.3 discusses using software performance engineering (SPE) techniques to estimate the system response times.

5.1 Modeling user performance

5.1.1 Cognitive engineering models

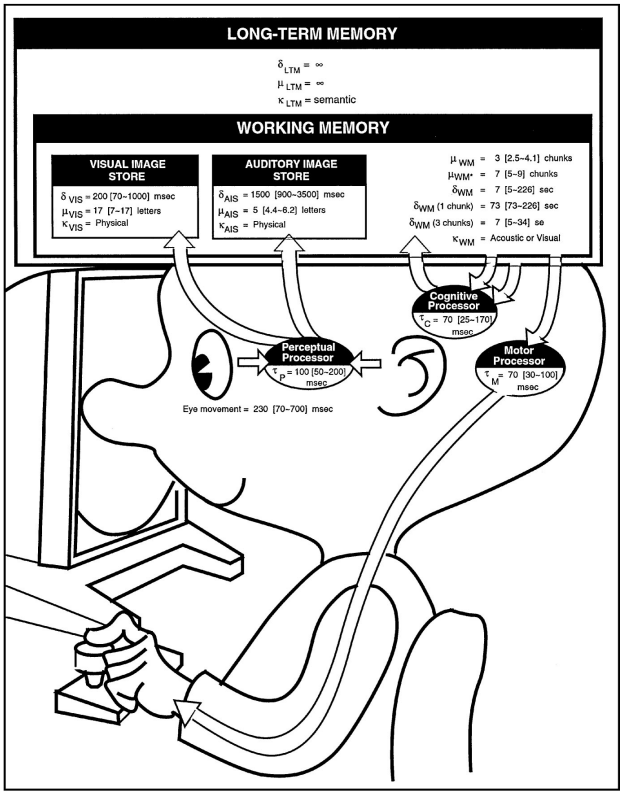
The nature of interaction between the human user and the computer is communication. Compared to the fast and diverse evolution of mobile systems, human users have evolved very little in basic capabilities from their ancestors. Users who interact with computers build up a skill set of efficient, smooth, learned behaviors for carrying out their routine communicative activities. The interaction process is intensely cognitive, even the most routine activities, such as using a computer text-editing program, require the interpretation of instructions, the formulation of sequence of commands, and the communication of these commands to the computer [26].

The theoretical basis of cognitive modeling methods used in this research is the *Model Human Processor* (MHP) described in the seminal book *The Psychology of Human-Computer Interaction* (CMN) [26]. An analogy to the information-processing system (i.e. computers) in terms of memories and processors, the MHP, which can be described by a set of memories and processors with a set of principles of operation, is intended to be used for making approximate predictions of human behavior. The MHP can be divided into three interacting systems: the *perceptual system,* the *motor system*, and the *cognitive system*, each with its own memories and processors, see Figure 5.1. For some tasks such as pressing a key in response to some text, the user behaves as a serial processor; for other tasks such as reading, it is possible for parallel operations of the three subsystems.

The MHP laid a foundation for HCI cognitive modeling research and practices to provide engineering models of human performance. Such models endeavor to predict execution time, learning time, and errors of human users and produce a priori quantitative predictions of performance at an earlier stage in the development process than prototyping and user testing. The predictions can be used to identify problems in a user interface as well as making comparisons among different design decisions. Ideally, these models should produce accurate predictions that are appropriate to different design situations, and they should allow designers without extensive training in psychology to use with minimal effort.

The quantitative predictability of cognitive engineering models is based on the extensive theoretical and empirical work by HCI researchers to estimate parameters that are robust and reliable across tasks. These parameters do not have to be fixed constants for all situations, but they must be determined a priori so as to be used without further validation to make predictions. These parameters incorporate psychological principles into the models, thus allowing computer engineers to use them without much psychological expertise.

Like all engineering models, cognitive models cannot cover the entire span of user computer interaction tasks, and there are issues like predicting creativity that may never be addressable with cognitive models. Fortunately, cognitive models can provide effective coverage of three extremely important issues [81]. First, the lower-level perceptual-motor issues, such as the effects of layout on key stroking or mouse pointing, can be captured by existing models. Second, the complexity and efficiency of the interface procedures is addressed very well by current models. Third, it is essential that how activities are performed together be considered for design.



Sensory information flows into Working Memory through the Perceptual Processor. Working Memory consists of activated chunks of Long-Term Memory. The Motor Processor is set in motion through activation of chunks in Working Memory. (From [26])

Figure 5.1: The Model Human Processor – memories and processors

Like all engineering models, cognitive models include only the details necessary to analyze the design, and are approximations to the processes involved in human behavior. The models allow designers to recognize when the design problem involves issues and factors not addressed by the models. The next sections will discuss new issues identified for mobile user interaction tasks.

Predicting user performance with cognitive models does not replace real user testing, rather, it should be used to reduce the amount of user testing required to improve usability. Cognitive modeling may be used together with other nonuser testing techniques [112, 115] during early design process to evaluate different designs and resolve potential design issues before investing in actual user testing and iterative development.

One fundamental principle of MHP on task analysis is the Rationality Principle [26], according to which, a user's behavior can be predicted by analyzing the task to determine the user's goals and operators with the constraints of the task. The GOMS model was created for taking into account the cognitive information-processing activities of the user.

5.1.2 The GOMS model

The GOMS model is specified by four components that form the user's cognitive structure: a set of **G**oals, a set of **O**perators, a set of **M**ethods for achieving the goals, and a set of **S**election rules for choosing among competing methods to achieve goals. Specifically:

Goals are what the user wants to accomplish by using the interface, the software, and the system. Goals are often divided into sub-goals, all of which must be accomplished to achieve the overall goal. Goals and sub-goals, if any, are often arranged hierarchically, but not required. This allows different levels of parallelism in variants of GOMS.

Operators are the actions that the system allows the user to take. Operators can change the user's internal mental state or physically change the state of the external environment. For command line user interfaces, an operator can be a command and its parameters typed on a keyboard. For GUIs, an operator can be a menu selection or a button press. For novel mobile user interfaces, an operator can be defined as a gesture stroke, a speech syllable, or an eye movement.

Methods are well-learned sequences of sub-goals and operators that can accomplish a goal. If a goal has a hierarchical form, there should be a corresponding hierarchy of methods. The content of the methods depends on the set of possible operators and the nature of the tasks.

Selection rules present the user's knowledge of which method should be applied if there is more than one method to accomplish the same goal. Selection rules can come from a user's personal experience or from explicit training.

Note that goals and operators differ at the required level of details. The analyst provides a method that uses operators to specify the details of how a specific goal is to be accomplished; in contrast, operators are usually more "primitive" and are not composed of any lower level operators.

The GOMS model has been widely known and verified by extensive HCI research [26, 61, 87, 114]. The parameters created for the original GOMS model have been extended to cover a wide range of tasks. Based on basic GOMS concept, there are four major versions of GOMS [80, 81]:

1. CMN-GOMS: This is the original formulation proposed in [26]. CMN-GOMS was a loosely defined demonstration of how to express a goal and sub-goals in a hierarchy of methods and operators and how to formulate selection rules.
2. NGOMSL (Natural GOMS Language) [87, 88]: This is a more rigorously defined version which presents a procedure for identifying all the GOMS components, expressed in a form similar to an ordinary computer programming language. NGOMSL includes rules-of-thumb about how many steps can be in a method, how goals are set and terminated, and what information needs to be remembered by the user while doing the task.
3. CPM-GOMS (Cognitive-Perceptual-Motor GOMS) [78]: This is a parallel-activity version that uses cognitive, perceptual, and motor operators in a critical-path method schedule chart (PERT chart) to show how activities can be performed in parallel.
4. KLM (Keystroke-Level Model) [25]: This is a simplified version that uses keystroke-level

operators (e.g. keystrokes and mouse movements) a user must perform to accomplish a task. A few heuristics are used in KLM to place "mental operators."

All four GOMS techniques produce quantitative and qualitative predictions of user performance, although each has different emphasis. GOMS models can be at several levels of analysis: the unit-task level, the functional level, the argument level, and the keystroke level. The research presented in this dissertation extends the keystroke-level modeling capabilities to not only user performance, but also energy consumption of mobile systems.

5.1.3 The Keystroke-Level Model

The KLM is based on a simple serial stage model of human information processing in which one activity is done at a time until the task is complete. In this model, all human information processing activities are assumed to be composed of primitive operators, including keystroke-level motor actions and internal perceptual and cognitive actions. To estimate task execution time, the analyst first specifies the method to accomplish a particular task, then lists the sequence of operators (which represents the method) and calculates the sum of execution times for all individual operators.

The original KLM [25] had four *physical-motor operators*: **K** (keystroking) represents pressing a key or a button, **P** (pointing) represents pointing with the mouse to a target on the display, **H** (homing) represents moving hands to the home position on the keyboard or mouse, and **D** (drawing) represents drawing lines using the mouse; one *mental operator*: **M** represents the mental preparation for a task; and one *system response operator*: **R** represents the system response time. For each physical-motor operator, the KLM gives an estimation of execution time, either a single value, a parametrized estimate, or a simple approximating function. For M, the KLM includes a set of heuristic rules for placing mental operators to account for mental preparation time during a task that requires several physical operators. For R, since different system process requires different response times, it must be estimated by the analyst as an input to the model, and is counted only if it causes the user to wait. The task execution time is thus the sum of the times spent executing the different operator types:

$$T_{execute} = T_K + T_P + T_H + T_D + T_M + T_P \tag{5.1}$$

where, for instance, the total time T_K spent in keystroking is the number of keystrokes $n_K$times the time per keystroke t_K, or $T_K = n_K t_K$.

The sequential architecture restricted the KLM to tasks that can be approximated by a series of operators, with no parallel activities, no interruptions, and no interleaving of goals. Like other GOMS models, the KLM predicts only error-free skilled behavior, and it does not predict the method to be used given the task situation. In addition, the KLM predicts only the time to execute a task, not the time to learn it. In summary, the KLM addresses the following prediction problem [25]:

Given: a task, which may involve several sub-tasks; the motor skill parameters of the user; the response time parameters of the system; and the method used for the task.

Predict: the time a skilled user will take to execute the task using the system with the given method without error.

5.1.4 Modeling parallel activities

While KLM is more suitable for tasks performed by a normally skilled user using sequential operations, there are other situations where extremely experienced user can perform subtle, overlapping patterns of activities, and as rapidly as the MHP permits.

The CPM-GOMS [78] is a parallel-activity version of GOMS that uses cognitive, perceptual, and motor operators in a critical-path method schedule chart (or PERT chart) to show how activities can be performed in parallel. A CPM-GOMS model of a user's task consists of boxes with durations and dependency lines between them. The critical path in a schedule chart provides the prediction of total task time. Much of the power of CPM-GOMS to predict skilled behavior comes from its ability to model overlapping actions by interleaving cognitive, perceptual, and motor operators. CPM-GOMS models are too detailed for tasks that can be usefully approximated by serial operators. CPM-GOMS models also make an assumption of extreme expertise in the user. That is, they typically model performance that has been optimized. When predicting user performance for mobile systems, CPM-GOMS can be used to complement KLM where parallelism needs to be addressed. Like KLM, the process of constructing CPM-GOMS can be automated [83].

5.2 Verifying KLM on pen-based interfaces

Initially, KLM was targeted mainly at text editing tasks on office desktop computers [25, 26]. Although it sometimes seen as a drawback of such models to assume error-free, skilled user interaction, the KLM has since then revealed remarkably precise prediction results in various applications such as email message organization [13], manual map digitizing [67], and vehicle navigation systems [105].

For mobile devices, most research on user performance has been limited to text entry for short messages [43, 119] and phone menu navigation [108]. However, other rich and novel user interaction techniques need to be taken into account. The serial stage model of the KLM ensures it useful in the paradigm of mobile user interaction, where most of the tasks are performed in an interactive, sequential manner. This section investigates the applicability and prediction accuracy of the KLM on pen-based, touchscreen mobile interfaces. The study first generated the KLM for four interactive tasks on a mobile device and predicted the task execution time. Then a user study was conducted to verify the predicted time. The results from comparing measured user time with predicted model time show that most original KLM operators can produce good prediction accuracy for touchscreen interfaces.

5.2.1 Task definition

Four interactive tasks were selected to verify the KLM on pen-based, touchscreen mobile interfaces. Two principles were kept in mind when choosing the tasks: first, the operations required to accomplish the tasks should cover as many different interaction methods used in the target platform as possible. Second, for comparison purposes, the tasks should be to accomplish the same goal but using different methods. The target mobile device used in this study is the Palm Vx PDA (Palm OS version 3.3 with 8MB RAM). The Palm Vx has a stylus pen, several hardware buttons, and a touch screen divided into two areas (Figure 5.2). The larger area on top is used to display information and allows the user to perform operations by tapping on icons, menus, lists, buttons, and other interface elements drawn on the screen. The smaller area at the bottom has four shortcut icons for quickly opening frequently used functions. The center of this area, known as *Graffiti*™, allows users to input text to the device by drawing shorthand gestures.

The target mobile application selected is an off-the-shelf software called ChoiceWay Guides (CWG) of New York City for Palm OS. This travel and city guide software allows the user to understand city facts, plan trips, and search for information like open hours and telephone numbers for a particular place. Figure 5.2(a) shows the start page of the CWG application. The user can tap on one of the icons displayed on the screen to perform corresponding operations indicated by the icon text.

All tasks in this study share the same goal of finding the opening hours of the Metropolitan Museum of Art (MET). Based on the functionality of the application, four different methods can be used to accomplish this goal:

Method 1: Map Navigation. From the start page shown in Figure 5.2(a), tapping on the "Maps" icon at the top right corner will lead to Figure 5.2(b), which displays the map of Manhattan divided into three regions. This method requires the user to have some basic knowledge of where the MET is in Manhattan. Tapping corresponding areas in the region map will lead to the detailed region map in Figure 5.2(c) and the street map in Figure 5.2(d). Tapping on the spot where the MET is located in the street map displays the name of the MET in a box at the bottom, which leads to the query result shown in Figure 5.2(f).

Method 2: Soft Keyboard. From the start page, tapping on the "Museums" icon located at the center right side gives Figure 5.2(e), an alphabetic list of museums. Using the soft keyboard located at the bottom of the screen, the user can then input the letters "METRO..." one by one. When MET is shown in the list, tapping on the item leads to the query result in Figure 5.2(f).

Method 3: Graffiti. The only difference from Method 2 is that at Figure 5.2(e) Graffiti is used to input the letters instead of the soft keyboard.

Method 4: Scroll Bar. From Figure 5.2(e) the user taps the scroll down arrow at the right of the list of museums until the MET is shown. The user then taps on MET to get the desired information.

(a) Start page (b) City map (c) Region map

(d) Street map (e) Museum list (f) Query result

Figure 5.2: Snapshots of the CWG for NYC application

5.2.2 Model creation

The KLM for the four tasks described above were created using an early version (v0_6) of CogTool [82], a suite of software tools built to facilitate modelers to quickly produce correct KLMs. CogTool v0_6 allows the analyst to mock up an interface as an HTML storyboard and demonstrate a task on the storyboard using the Netscape web browser. Chapter 6 will further present modeling work using a more recent version of CogTool, which no longer uses HTML. The demonstration events are captured by the Behavior Recorder [90] module of CogTool, which automatically generates a KLM that includes all Ks, Ps, Hs, and Ms required to accomplish the task. The KLM is implemented in ACT-Simple [131] which compiles into ACT-R [12] code. The task execution time is then calculated by running the generated KLMs in the ACT-R environment.

The HTML mock-ups for the four tasks were generated from the Palm OS Emulator, which

```
(klm-p (klm-goal klm
(think)
(look-at "Museums")
(press-button "Museums")
(think)
(look-at "-graffiti-")
(press-button "-graffiti-")
(think)
(press-button)
(think)
(look-at "MET")
(press-button "MET")
...  ...
```

Figure 5.3: Example KLM code for Method 3 - Graffiti

emulates the hardware of various models of Palm handhelds. The emulator enables a "virtual" handheld device to run on a desktop machine. The CWG application was installed on the emulated Palm Vx, and the snapshots of each step taken in tasks were taken. The pictures in Figure 5.2 are examples of the snapshots. The snapshots were then used to create the HTML mock-ups. More details about using CogTool can be found in [82, 90]. Figure 5.3 shows a fraction of the KLM generated for the Graffiti task, expressed in ACT-Simple code. In this model, the physical-motor operators such as `(press-button Museums)` were captured by the CogTool Behavior Recorder when the task was demonstrated. The `(look-at)` and `(think)` operators were automatically added by CogTool. Task execution times were calculated by running the KLMs in the ACT-R environment.

5.2.3 User study

To verify the task execution time predicted by the KLMs, a study was conducted with 10 skilled PDA users. All the participants were college or graduate students who own one of the several kinds of handheld devices: Palm OS or Pocket PC PDAs, or smart phones. Although not all of these PDA users were skilled at Graffiti, they all were skilled at gesture-based text entry and the training session (described later in this section) allowed them to get familiar with the three Graffiti gestures required for Method 3. Figure 5.4 shows the information of each participant including the gender, the model of PDA owned, and for how long it has been used.

User task execution times were obtained using EventLogger, a Palm OS system extension that records system events to a log file. The log files are Palm database (PDB) files in text format. Each line of a PDB log file is a tab-delimited listing of one system event, in the form of "*TickCount sysEventName OptionalInfo*". The *TickCount* is the time stamp of the event, the *sysEventName* is the name of the event, and the *OptionalInfo* includes information such as the character entered in a keystroke event, the name of the form in a form open event, etc. User execution time for each task can be obtained by calculating the difference between the starting and ending event timestamp, and dividing the difference by the number of system ticks per

User #	Gender	Device owned (OS)	Time
1	Male	Palm Vx (Palm OS)	5 years
2	Male	Palm IIIe (Palm OS)	4 years
3	Female	Palm VA (Palm OS)	3 years
4	Male	Handspring Visor (Palm OS)	3 years
5	Female	Handspring visor Pro (Palm OS)	2 years
6	Male	Kyocera 7135 (Palm OS)	4 years
7	Male	Handspring Visor Prism (Palm OS)	3 years
8	Male	Compaq iPAQ (Win CE)	3 years
9	Female	Dell PDA (Win CE)	1 year
10	Male	iPAQ 3630 (Win CE)	4 years

Figure 5.4: Participants device usage

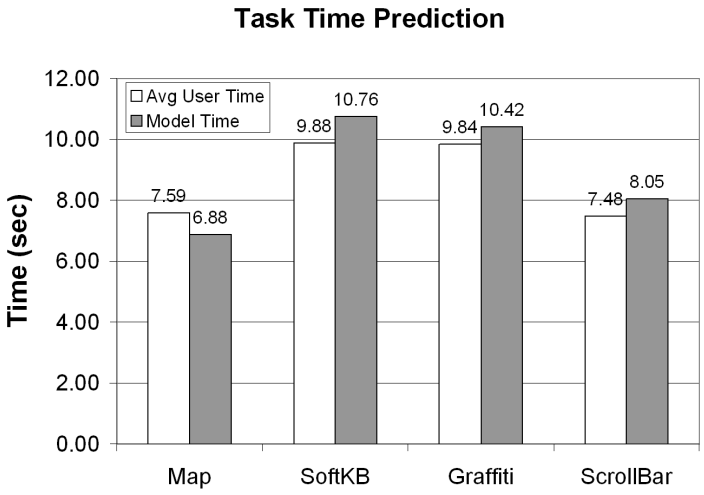
second defined in the header part of the PDB file.

Each participant was first asked to practice the tasks in a training session, followed by the actual session where the participants were asked to perform the four tasks 10 times each. In the training session, participants were asked to carefully read the step-by-step instructions on how to operate the EventLogger and how to perform the tasks. The participants were required to strictly follow all steps and repeat each task 10 times. During this training session, the participants were told to focus on becoming familiar with the tasks. The PDB files from this session were saved as training data. In the second session, the participants were asked to run each task for 10 times again without referring to the instruction, assuming they had all become familiar with the tasks during the training. In total, we collected 400 user execution log files from the second session. 20 files were not usable because the user forgot to or did not start the EventLogger correctly, these files were thrown away.

5.2.4 Result analysis

Figure 5.5(b) lists the result of the user study including the average, maximum, and minimum task execution time and standard deviation. It also lists the model time and the prediction error for each task, and the average prediction error is 7.9%, which is consistent with the 6% average error rate reported in [82]. However, because all the four tasks under study are very short, small discrepancy between model time and user time can result in significant prediction errors. The models underpredict the Map Navigation method on one design, and overpredict the other three methods on the other design. This discrepancy can be attributed to the difference in design and system response time estimation. Figure 5.5(a) illustrates the comparison of model predicted time and measured average user time.

The standard deviations of the Map Navigation and the Graffiti tasks are higher than the other two tasks. This may contribute to the fact that using the software keyboard or scroll bar to find and select an item from a list is a task that most users are very familiar with. By comparison,



(a)

Task	User time (sec)			Std. dev.	Model time (sec)	Diff (sec)	Error
	Average	Max	Min				
Map Navigation	7.59	10.44	6.00	1.36	6.88	0.71	9.3%
Software Keyboard	9.88	11.39	9.20	0.79	10.76	0.88	-8.9%
Graffiti	9.84	12.53	8.34	1.63	10.42	0.56	-5.8%
Scroll Bar	7.48	8.31	6.54	0.62	8.05	0.56	-7.7%

(b)

Figure 5.5: Task execution time: predicted versus measured

the variation in users' familiarity with the location of MET when navigating a map, as well as in their expertise of using Graffiti strokes is higher, which results in higher difference in user time.

The modeling process and resulting predictions have revealed two important issues that had not been addressed before. First, the stylus-based interface on mobile devices introduces the need to update the original KLM parameters and rules. During the early stage of this study, it is found that *Graffiti stroke* should be added as a new operator to address the time for the user to make handwriting gestures and for the system to recognize the gesture. A value of 580ms is used for each Graffiti stroke, based on a previous study [48]. Secondly, to accurately predicting total task time, the value of the system response operator R is very important due to the comparatively lower processing speed of mobile systems. In this study, because both the software application and the mobile platform already exist, it was easy to identify where during each task the user had to wait for the system and to input the corresponding $R(t)$ for modeling, where t is the response time parameter of R [25] . For example, the estimated system response time to load the museum list was 4200 ms, and the time to update the list was 2300 ms.

5.3 Estimate system response time

The accuracy of KLM prediction can be greatly affected if the system response time operator R is not carefully estimated. Because response times are determined by software implementation and underlying systems, the KLM does not embody a theory of system response time, which must be input to the model by giving specific values for the parameter *t*. System response times can overlap with mental operators such as task acquisition. Only the non-overlapping portion of the response time is counted in the total task time, therefore the actual user wait time is not necessarily the same as the time required by the system.

In many cases, the system response time is negligible because of the great speed disparity between human and computer; in other cases, however, such response delay can be substantial, especially in the context of mobile devices with lower computing capacity and slower connectivity. Two good examples are database retrieval and web page loading. In such cases, the system response time is determined by the typical user requests and the amount of processing they require. For instance, a database query will be slow if it causes many complex record qualifications and I/O operations, but fast if it consists of a few accesses to already filled system buffers. If the small, fast request is typical, the average responsiveness as seen by the users will be good. Responsiveness is also determined by the computer system resources available.

In the literature of using the KLM to estimate task execution time, it may not be necessary to include the R operator, either because it takes near-zero time, or it has the same value in all of the alternative designs, and so affects only the absolute, not the relative task times. However, to achieve accurate system energy prediction, quantitative characterizations of the system activities invoked by user interaction must be obtained. In this research, software performance engineering (SPE) techniques are used to estimate the system response times. This section briefly describes the current practices of SPE, in the next chapter, SPE methods will be implemented in the process of assessing energy consumption.

Software Performance Engineering (SPE) is a method for constructing software systems to meet performance objectives [148, 149, 150]. The SPE process begins early in the software life cycle and uses quantitative methods to identify satisfactory designs and to eliminate those that are likely to have unacceptable performance, before developers invest significant time in implementation.

To ensure that early performance modeling overcomes the lack of knowledge about the software design and implementation, SPE suggests using software execution models to get rapid evaluation on software performance by providing a static analysis of the mean/expected, best- and worst-case response times. The data required during early design to construct quantitative software execution models include workload scenarios, software design concept, execution environment, and resource usage estimates. At later phases when more details about design and implementation are known, the estimates will become more precise.

Software execution models are expressed in *execution graphs*, which provide a visual representation of the software processing steps (similar to UML activity diagrams) for a specific workload scenario. Figure 5.6 gives some basic notations of software execution graphs. The graphs consists of nodes (represent processing steps) and arcs (represents the order of execu-

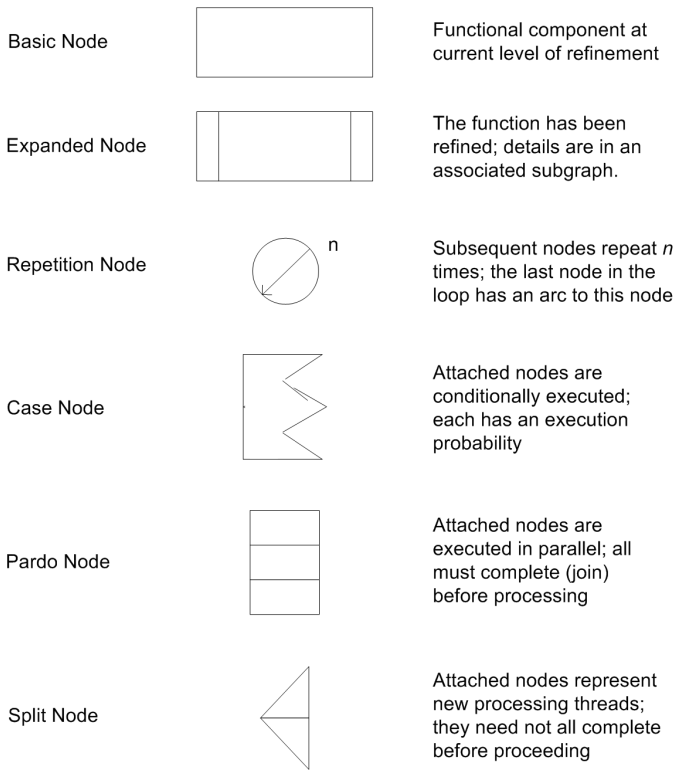


Figure 5.6: Basic notations of software execution graphs (From [149])

tion). Figure 5.7 shows an example execution graph of the Map Navigation task.

The processing steps in an execution graph can be described in terms of *software resources*, which capture computational needs that are meaningful from a software perspective. For example, the number of database accesses or size of data transmission required in a processing step may be specified. Software resources depend on the type of application and the operating environment. The types of software resources that are important for the CWG application are screens (the number of screens displayed to the user) and database accesses (the number of database retrieval).

For each software resource request, the *system resource requirements* must also be specified. These requirements connect software resource requirements to hardware usage in the target environment, as well as specify characteristics of the operating environment such as the speed of the processor. A *frame of reference* for estimating resource usage is necessary, i.e., if the resource requirements for similar activities are known, it is easy to extrapolate to new software. The next chapter will describe the practices in obtaining resource estimations for energy modeling.

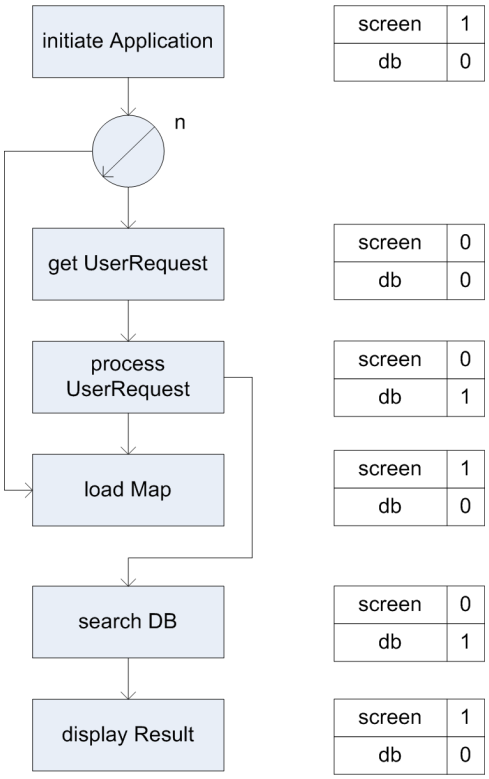


Figure 5.7: An example execution graph of the Map Navigation task

5.4 Summary

This chapter has described using cognitive modeling techniques to model human user performance when interacting with mobile devices. The GOMS techniques is one of the most widely known theoretical concepts for studying the efficiency of user interaction. Because KLM addresses keystroke-level, serial interactive operations of the user, it is suitable to be used in modeling mobile user interaction tasks by nature. This chapter investigates the predictability of KLM on a pen-based mobile user interface, and the experimental results show similar predicted error to those reported for KLM in the literature. Furthermore, this chapter suggests using software performance engineering techniques to estimate the system response time parameter for the KLM.

Chapter 6

Keystroke Level Energy Modeling

This chapter presents the Keystroke-Level Energy Model (KLEM), a quantitative analysis methodology that predicts both user performance and system energy consumption of an interactive task, during early design phases. The KLEM extends the KLM, and integrates system energy consumption with the user interaction model of the KLM, thus predicts both system energy consumption and user interaction time.

6.1 Extending KLM for energy prediction

Chapter 5 has shown that the KLM provides a good granularity of details to describe tasks in the context of mobile computing. However, as the KLM only predicts task execution time, more information of the system should be added in the model to serve the need of energy prediction.

6.1.1 Overview of KLEM

The total energy consumption of a mobile system during an interactive task is mainly decided by two factors: the level of system power consumption, and the time spent on each power level. Let S be the set of system power states of a task, P_s be the power level of each state, T_s be the time the system stays in state s, then the task energy consumption E_t is:

$$E_t = \sum_{s \in S} P_s T_s \tag{6.1}$$

Therefore, to obtain accurate predictions on task energy consumption, the model should be able to describe a task as a series of activities. During each activity the system is in a certain power state s for time T_s. The better the activities and their corresponding system power states are defined in the model, the more accurate the prediction will be.

In the original KLM, a task is described by listing the sequence of operators that include elementary perceptual, and motor or cognitive actions of the human user. Given a task, the methods used to accomplish the task, the proposed interface design, and a target platform, KLEM

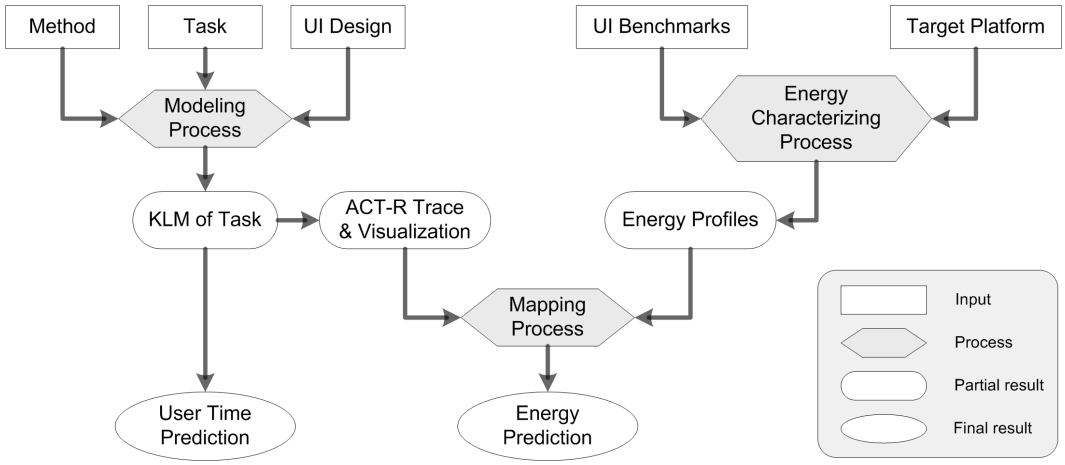


Figure 6.1: The process of constructing KLEM

extends the original KLM prediction of the error-free task time of a skilled user, and predicts the system energy consumption during the task execution. As defined in KLM, a task is a series of interactive operations a user performs on a computer system to achieve a certain goal, e.g., schedule a meeting, inquire about store hours, or changing the system settings.

Figure 6.1 depicts the components and processes of the KLEM technique. The resulting model predicts task execution time (User Time Prediction) and task energy (Energy Prediction). The task execution time is obtained by running the *Modeling Process*, which basically constructs a KLM for each task. Specifically, the KLM is represented by the underlying computation cognitive engine of CogTool as a ACT-R [2] Trace with a model Visualization that illustrates what ACT-R is doing. The *Energy Characterizing Process* obtains the necessary Energy Profiles by running a set of UI Benchmarks on the Target Platform. The ACT-R Trace and Visualization obtained from the Modeling Process are then used in the *Mapping Process*, in which the KLM of Task and the Energy Profiles are joined to produce Energy Prediction of the task. Therefore, the Modeling Process focuses on constructing the set of T_s, the Energy Characterizing Process focuses on constructing the set of power states S and obtaining the value of P_s, and the Mapping Process produces E_T in Equation 6.1.

6.1.2 Modeling process

The first step is to construct the KLM of a given task during the *Modeling Process*. This process focuses on profiling T_s in Equation 6.1, which should not only include the model user's operations time, but also contain information on the system activities corresponding to these user operations during the task. A newer version of CogTool [3] is used to generate the KLM of skilled user performance on the tasks under study. ACT-R Traces and model Visualizations are generated by demonstrating each task on its corresponding design storyboards. In addition to touchscreen and Graffiti interfaces, this version of CogTool also supports modeling auditory

interfaces using the HEAR and SAY operators.

To create a storyboard of the proposed user interface design, a series of frames that represent the display changes caused by user operations are needed. Although the screenshots of acutal applications are used in this study, in reality each frame can be as simple as a sketch of the proposed interface design. The basic visual building blocks of a GUI storyboard are widgets, such as a button or a menu, which provide a point of interaction for the user to manipulate the data associated with the widget. In CogTool, a Widget is represented as a "hot spot" in a frame to indicate an interactive area on the actual physical device. Interactive widgets that are currently supported by CogTool include Button, Check box, Radio button, Textbox, Pull-down list, List box, Menu (including header, submenu, and menu item), and handwriting input area (e.g. Graffiti™ in Palm OS and Soft Input Panel (SIP) in Windows Mobile). In CogTool, widgets can also represent other interactive elements, such as hardware buttons or cursor devices like joysticks. These elements can also be represented as hot spots on the storyboard. To support auditory or speech-based interactions, CogTool has SAY transitions, and can model the cognitive time of HEAR. For auditory interfaces without a display, storyboards and frames are still needed for modeling the transitions of system states caused by auditory inputs, but the storyboard does not necessarily contain GUI elements.

After creating the storyboard, the set of tasks that will be executed on the proposed interface needs to be defined and demonstrated on the storyboard. The demonstration simulates the steps a user would perform to accomplish a task and is recorded to a script for CogTool to generate prediction on total task time and the corresponding ACT-R trace. Figure 6.2 shows an example of a portion of ACT-R trace generated by CogTool. Each trace contains the user's perceptual, motor, and cognitive activities, as well as the necessary system responding activities that are provided by the modeler.

For instance, in Figure 6.2, at `0.683` second the model fires a **`MOTOR`** operation, which is a tap on the touchscreen at location `(278.0 177.0)`. This user operation (the tap) is a trigger to a GUI event that causes the model storyboard (the device interface) to start transitioning to the next frame (change display content). The computer system needs to handle this event and produce responses to this user operation, and therefore consumes time and energy. In the example trace, at `0.683` second, the **`MOTOR`** operator causes the storyboard to transition to the next frame, which represents a display update in the computer system. Similarly, at line `0.768` **`PROCEDURAL PRODUCTION-SELECTED WAIT-FOR-SYSTEM-5`**, the model user starts waiting for the system until the system restores display at `1.324` second. This duration of waiting is the system response time operator in KLM and needs to be carefully estimated, as discussed in Section 5.3. The system response time can be caused by any sort of computation or communication job that the system must finish before the user can continue operation. The **`PROCEDUAL`** operators during the system response time represent the cognitive processes of the model during the task and the **`VISION`** operators represent various visual preparation processes.

```
...
0.400 MOTOR INITIATION-COMPLETE
0.400 PROCEDURAL CONFLICT-RESOLUTION
0.683 MOTOR MOVE-CURSOR-ABSOLUTE #(278.0 177.0)
0.683 Storyboard transitioning to frame "List1"
0.683 PROCEDURAL CONFLICT-RESOLUTION
0.733 MOTOR FINISH-MOVEMENT
0.733 PROCEDURAL CONFLICT-RESOLUTION
0.768 VISION Encoding-complete LOC1-0 NIL
0.768 PROCEDURAL PRODUCTION-SELECTED WAIT-FOR-SYSTEM-5
0.768 PROCEDURAL BUFFER-READ-ACTION GOAL
...
0.768 PROCEDURAL BUFFER-READ-ACTION GOAL
0.768 PROCEDURAL QUERY-BUFFER-ACTION MANUAL
0.818 VISION CHANGE-STATE LAST NONE PREP FREE
1.324 COGTOOL Restoring display at end of system wait (0.556)
...
```

Figure 6.2: An example ACT-R model trace created using CogTool

6.1.3 Energy characterizing process

KLEM enables early estimation and comparison of the energy consumption of different designs by taking a black-box approach in the *Energy Characterizing Process*. User interface designs are often separated from the system design on power optimization and management, and lower level energy characterization is often not available at the level of UI design. KLEM bridges this gap between the design of the UI and lower levels of the system. Because of the highly interactive nature of mobile tasks, it is more beneficial to identify and solve potential energy problems at a higher level of the system, and at an early stage of the entire produce life cycle.

In the Energy Characterizing Process, a measurement-based approach is used to obtain the energy profiles of KLEM operators by running a set of benchmarks on the target platform. This approach is suitable when the target platform is already available at the time of application design, and is easy to reproduce on any mobile platform as little platform-specific software instrumentation is required to run the benchmarks. If the target platform is not available for benchmarking, energy profiles can be obtained from manufacturer hardware datasheets or hardware power consumption characterizations from literature.

Based on keystroke level operations, the Energy Characterizing Process profiles the power and energy consumption of interactive tasks that are performed by a user on a mobile platform. Therefore the assumptions behind this approach are that at any time, the user only performs one task, and the current task is the only energy consuming application in the system without concurrency. The approach also assumes that the system does not enter low-power sleep mode or employ dynamic energy management mechnisms (e.g. voltage scaling) during the tasks. From this perspective, KLM is a very suitable basis for energy characterization extensions.

Widget	Operation	System Activity
Selection		
Button	Tap	Small, Medium, Large
Checkbox	Tap	Small
List box	Tap	Small
Dropdown list	Tap + Tap	Small
Radio button	Tap	Small
Menu	Tap	Small, Medium, Large
Hardware Button	Tap	Small, Medium, Large
Navigation		
Tab	Tap	Medium, Large
Scrollbar	Tap/Drag	Small, Medium, Large
Slider	Tap/Drag	Small
Text Input		
Soft keyboard	Tap	Small
Handwriting	Stroke	Medium, Large
Speech Interaction		
Hear	Hear	Medium, Large
Say	Say	Medium, Large

Figure 6.3: Interaction activity benchmarks

Interaction benchmarks

As addressed in Equation 6.1, the energy characterizing process is twofold. To characterize T_s, a set of interaction benchmarks is created for capturing the activities performed on common interactive widgets. Figure 6.3 lists the benchmarks grouped by similar functions: Selection, Navigation, Text Input, and Speech Interaction. The operation(s) the user can perform on each widget and the corresponding amount of system activities, both in time and in energy, are also listed.

Usually, buttons provide the quickest access to functions provided by the interface, but they occupy a fair amount of screen space, and having too many buttons on a form is inefficient because novice users must spend more time visually searching the screen for the button they want. Widgets that require one tap to make a selection are faster than widgets that require two taps to make a selection. A list is faster than Graffiti or on-screen keyboard input since the user will spend less time entering data. Lists that contain a lot of elements are slower to use than short lists because it is difficult to take a glance at too many list items and the user may have to scroll the list to find the right item, requiring yet another tap. Menus require an extra tap to display the menu in the first place, and should be reserved for functions that are less frequently used.

In the Selection group, the widgets Checkbox, List box, Dropdown list, and Radio button are usually used to make a selection among several items that the user operates by tapping the

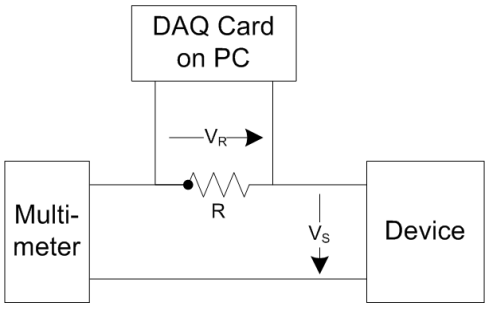


Figure 6.4: Power measurement testbed

widget. Note that the Dropdown list widget requires two taps to perform a selection, and one can use tapping and dragging to operate the Scrollbar and Slider widgets. After the user selects an item, the application records the selection, updates a small area of the display to look responsive to user operation, and goes back to the waiting/idling state for the next user operation. This kind of system activity is considered "Small". As for "Medium" and "Large" system activities, for instance, if a "next" button is pressed to open a new window, the consequent system activity is defined "Medium", while an "open" button that reads a 1MB file is considered a "Large" system activity.

Power measurement setup

The testbed used to obtain the energy profiles as well as the energy measurements for model verification is shown in Figure 6.4. The battery is removed from the device under measurement to eliminate the current draw due to battery charging. The device was connected directly to the external power supply and the input current I was obtained by measuring the voltage V_R across a 1 Ohm resistor R connected in series with the device, and $I = \frac{V_R}{R}$. The voltage value is sampled at 10 KHz using a high speed Data Acquisition Card (DAQ). Minor fluctuations are ignored in the supply voltage V_s, which is assumed to be constant. The system power consumption is then calculated as $P = V_s I = V_s \frac{V_R}{R}$.

The voltage samples that the high speed DAQ collected during each task are stored in a .dat file and can be converted into a .txt file that is readable to MATLAB, using which the voltage samples are processed, plotted, and analyzed to produce the power state machines and energy profiles for different user operations.

Power state machines

Based on Equation 6.1, the energy consumption of a KLEM operator is defined as the sum of the KLM operator energy and the system activity energy it invoked. The power states of most of the tasks studied in this chapter can be expressed using a simple state machine showin in Figure 6.5. During these tasks, the system activity alters between two states: Idle and Busy, with corresponding power levels P_i and P_b.

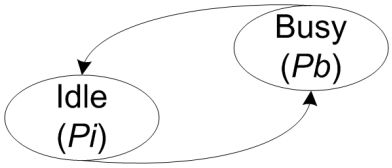


Figure 6.5: A power state machine of typical graphical interface

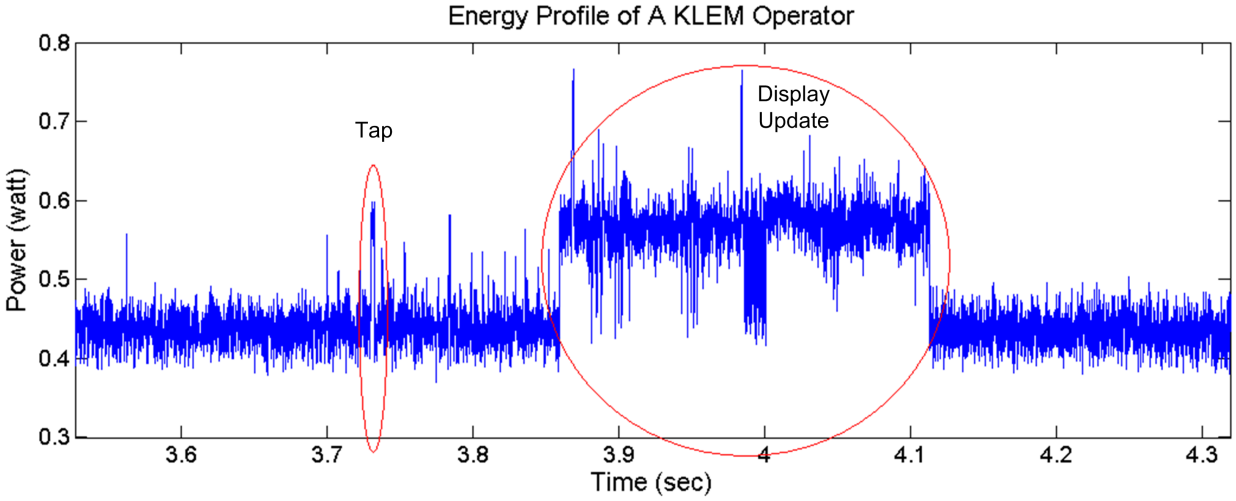


Figure 6.6: Energy profile of a button press followed by a display update

The value of each power state is obtained from running the interaction activity benchmarks on the target platform. Figure 6.6 depicts the energy profile of a "button press" (Tap) operation followed by a full display update (e.g. open a new window, load a picture) measured on one of the target platforms studied. As an example, the Figure shows the raw trace of power samples obtained during one measurement of the button press benchmark. The power parameters that will be used in the model are based on the average power value over 10 measurements of each operation. The values along the y axis are the instantaneous power level at time t, and the system energy consumption during any time interval (t_1, t_2) is the area under the power trace between t_1 and t_2. In Figure 6.6, the oval on the left indicates the energy profile of the Tap operation and the circle on the right indicates the energy profile of the display update activity. The lower power level (e.g. 3.6 second to 3.7 second) corresponds to P_i in the Idle state, and the higher power level (e.g. 3.9 second to 4.1 second) corresponds to P_b in the Busy state.

A more complex example of power state machine and energy profile – the one for a speech recognition interface that does a simple job of searching for a certain name in the contact list – is shown in Figure 6.7 and Figure 6.8, respectively. Upon running, the speech interface enters the Busy state, which corresponds to approximately 2 second to 3 second in the energy profile. It then plays a voice prompt "speak now" (at around 3 second), and enters the Speech Preprocessing state (3.2 second to 3.8 second) where the interface awaits the user to speak something.

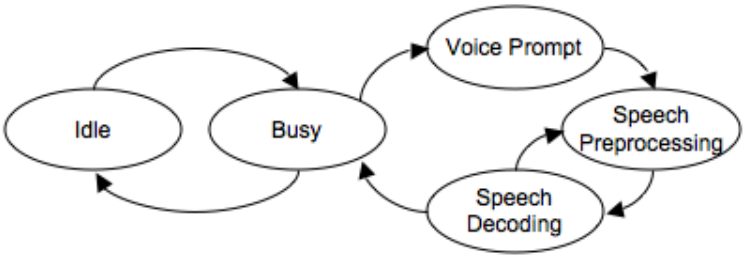


Figure 6.7: Power state machine of a speech interface

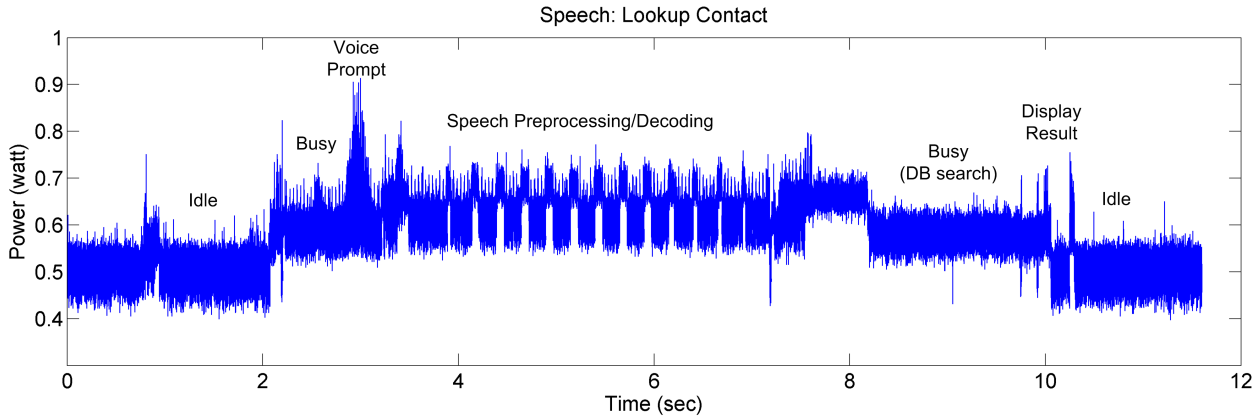


Figure 6.8: Energy profile of a speech recognition task

Between 3.5 second to 7 second, the user speaks the sentence "look up contact John Reed" and the interface transits between the Speech Preprocessing state and the Speech Decoding state to decode the audio input and recognize the spoken sentence. At around 7.5 second to 8.2 second, the interface finishes the speech-to-text recognition, activates the searching functionality to query the information of John Reed in the contact list, and displays the query result on screen. Then at around 10 second, the interface goes back to Idle and waits for the next speech input.

To obtain the energy profiles of possible power states a target platform can be in, a comprehensive set of benchmarks should be defined. A typical set of such power benchmarks are listed in Figure 6.9. As stated before, the energy characterizing process of KLEM takes a black-box approach and the modeling is made during design time, when there is no application implemented for obtaining the power values. The solution is to run a set of off-the-shelf applications as the benchmark and obtain the power levels of different system activities. After the implementation stage, the actually measured power levels of some activities, for instance Speech Decoding, may be different than the benchmarks, but as is stated in Section 5.3, it is more important to focus on the relative energy value of different design options, and the absolute values can be easily updated later in the implementation cycle.

In Figure 6.9, the left column lists the benchmarks for obtaining the necessary power profiles of typical system activities involving different hardware components, and the right column gives

Benchmark	**Energy Calculation**
Idle	$T \times P_{idle}$
Busy (computing)	$T \times P_{busy}$
Idle no LCD	$T \times P_{idlenoLCD}$
Busy no LCD (computing)	$T \times P_{busynoLCD}$
Backlight (level 0 to 10)	$T \times (P_{level} + P_{state})$
Display only (panel + backlight)	N/A
Media Player	$T \times C \times P_{audiobase}$
Media Player no LCD	$T \times C \times P_{audiobase}$
Voice Prompt	$\frac{N}{W} \times C \times P_{audiobase}$
Speech Preprocessing	$\frac{N}{W} \times C \times P_{sp}$
Speech Decoding	$\frac{N}{W} \times C \times P_{sd}$
Voice Prompt no LCD	$\frac{N}{W} \times C \times P_{audiobasenoLCD}$
Speech Preprocessing no LCD	$\frac{N}{W} \times C \times P_{spnoLCD}$
Speech Decoding no LCD	$\frac{N}{W} \times C \times P_{sdnoLCD}$
Text to Speech	$\frac{N}{W} \times C \times P_{ttsbase}$
Text to Speech WiFi Idle (beacon every 100 ms)	$T \times P_{wifiidle}$
WiFi Active Transmit	$\frac{D}{S} \times P_{wifitv}$
WiFi Active Receive	$\frac{D}{S} \times P_{wifirv}$

P_{state}: current power state C: volume coefficient
N: number of words W: words per second
D: data to transfer (bit) S: throughput (bps)

Figure 6.9: Example benchmarks to obtain typical system power states

the corresponding energy model for calculating the energy consumption of each power state. For instance, for system energy consumption in different backlight levels, the power benchmark in the fifth row – Backlight (level 0 to 10) – should be used, and the energy is calculated using $T \times (P_{level} + P_{state})$, where T is the duration the system is in this power state, P_{level} is the additional power consumption of each backlight level, and P_{state} is the current power state the system is in. For the same system activity, the higher the backlight level, the more the power consumption.

For more complex system activities that involve speech recognition or network transmission, the energy calculation formula not only depends on hardware power levels, but also requires detailed information about the data being recognized/transmitted to determine the system response time. Analysis methods on system response time have been discussed in Section 5.3.

6.1.4 Mapping process

The Mapping Process takes the predicted total task time T_{task}, the storyboard and ACT-R trace that contain contextual information of system activities, and the energy profiles obtained in the Energy Characterizing Process, and produces the task energy prediction. Let O be the sequence

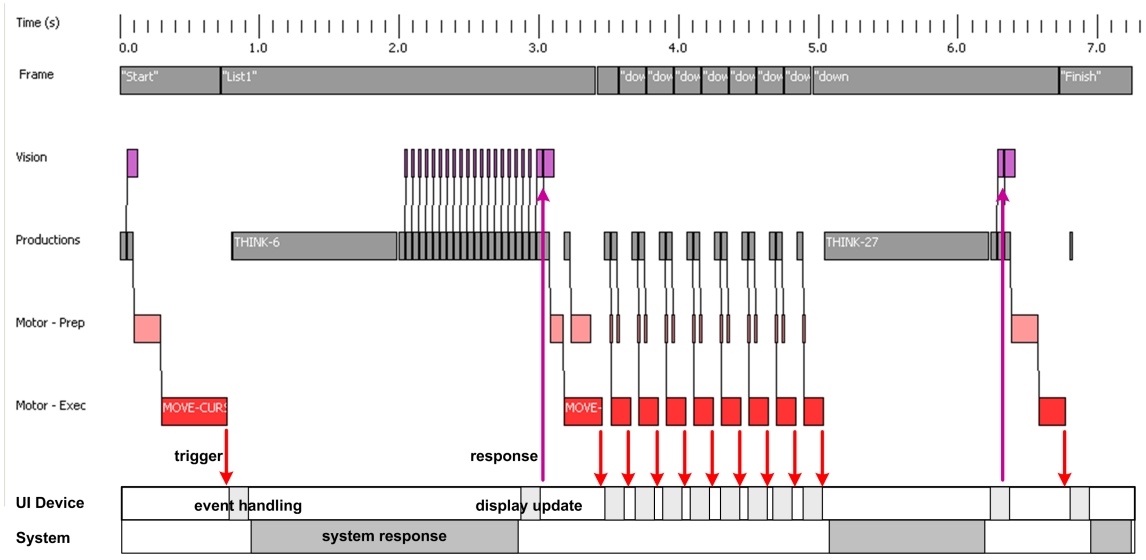


Figure 6.10: Obtaining system activity semantics from model visualization

of KLEM operators in the task, T_o be the time of operator o, the total system energy consumption during idle state E_{idle} is:

$$E_{idle} = P_i(T_{task} - \sum_{o \in O} T_o) \tag{6.2}$$

The total task energy can be predicted using:

$$E_{task} = \sum_{o \in O} E_o + E_{idle} \tag{6.3}$$

The actual Mapping Process needs more effort than described in Equations 6.2 and 6.3 above. In CogTool, the underlying ACT-R computation cognitive engine that makes the predictions is very complex, and so is reading the model trace directly. A visualization tool is built into CogTool to help the designer to see what ACT-R is doing to produce the predictions. The upper part (indicated by the first six rows labeled **`Time(s)`**, **`Frame`**, **`Vision`**, **`Productions`**, **`Motor-Prep`**, and **`Motor-Exec`**, correspondingly) of Figure 6.10 is the visualization of a short task of tapping the "down" arrow of a scroll bar to browse the contents in the scroll list. The **`Time(s)`** row is a timeline showing the different activities ACT-R goes through to make the predictions. The **`Frame`** row shows the duration that each frame in the storyboard is visible. Change from a frame to another in the storyboard often corresponds to a screen update in a GUI.

The lower rows of boxes are different types of operators that happen in the course of performing the task. The **`Vision`** row represent the cognitive operations of eyes seeking objects in the frame. The **`Productions`** row represent the thoughts the model has when performing this task. The longer boxes are "Think" operators and the shorter boxes are other types of cognitive operators that indicate motor movements and visual attention shifts. The **`Motor-Prep`** and

Motor-Exec rows represent different aspects of the motor system, and the **Motor-Exec** row shows observable motor movements of a finger, stylus, or button press.

When mapping an ACT-R trace to produce an energy prediction, the activities that can cause non-idle system activities are the **MOTOR** and **WAIT-FOR-SYSTEM** operators in Figure 6.2, and system activities result changes in power states. During other activities such as the **VISION** and **PROCEDUAL** the system is usually idle waiting for user operations. The energy consumption of non-idle system activities not only depends on the hardware platform, operating system, and application software, it also depends on the particular interaction method. For instance, a **MOTOR** activity can be tapping, dragging, key pressing or releasing, or a handwriting stroke. In this research, the system activities associated with each operator need to be manually identified from the model. Future modeling tools should allow integrating the information of system activities and power profiles into the model for automatic energy calculation.

The task being modeled in Figure 6.10 contains only "tapping" operations, therefore the **Motor-Exec** row represent taps on the touch screen, which trigger UI event handling and system activities discussed in previous sections. When the system finishes processing, the corresponding response in turn triggers **Vision** operators in the model user, who takes further actions to tap the screen.

If all motor operators are correctly captured for a task, as is depicted in Figure 6.11, where the Motor operators are taps, the model visualization can be a good profiler of actual energy consumption. As ACT-R cannot do "drag and drop" yet, CogTool has the inherited weakness of modeling a dragging operation. Currently CogTool approximates drag-and-drop with hover-and-click, since both operations have one down-press, one release, and one movement. Although the time prediction might be very close to the actual measurement, the model visualization does not necessarily reflect the actual energy profile because the constant display-update system activity during the movement is not captured in the model. This discrepancy causes a “poor” mapping shown in Figure 6.12. Here the “poor” mapping does not mean poor energy prediction, because the model can still accurately predict the time of user operations that trigger system activities. Modelers need to perform manual analysis of the relationship between these user operations and corresponding system activities.

6.2 Verification

Two PDA platforms, an iPaq RX1955 and a Tungsten T5 are used to verify the prediction accuracy of KLEM on user time and system energy. The specifications of devices are summarized in Figure 6.13.

Two principles are applied when choosing the tasks to validate the KLEM model. First, the operations required to accomplish the tasks should cover as many different interaction methods available in the target platforms as possible. Second, for comparison purposes, the same goal should be accomplishable by using different interaction methods. The same off-the-shelf tour guide application ChoiceWay Guides (CWG) for New York City is used because it has releases for both Windows Mobile and Palm OS for comparison purposes.

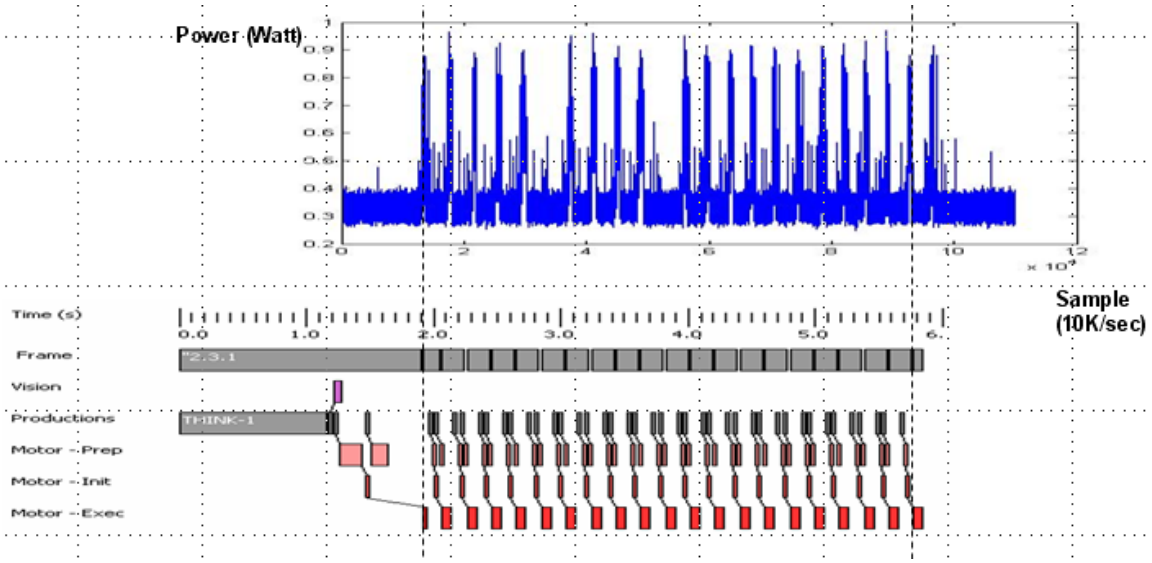


Figure 6.11: A good mapping of energy profile and model visualization

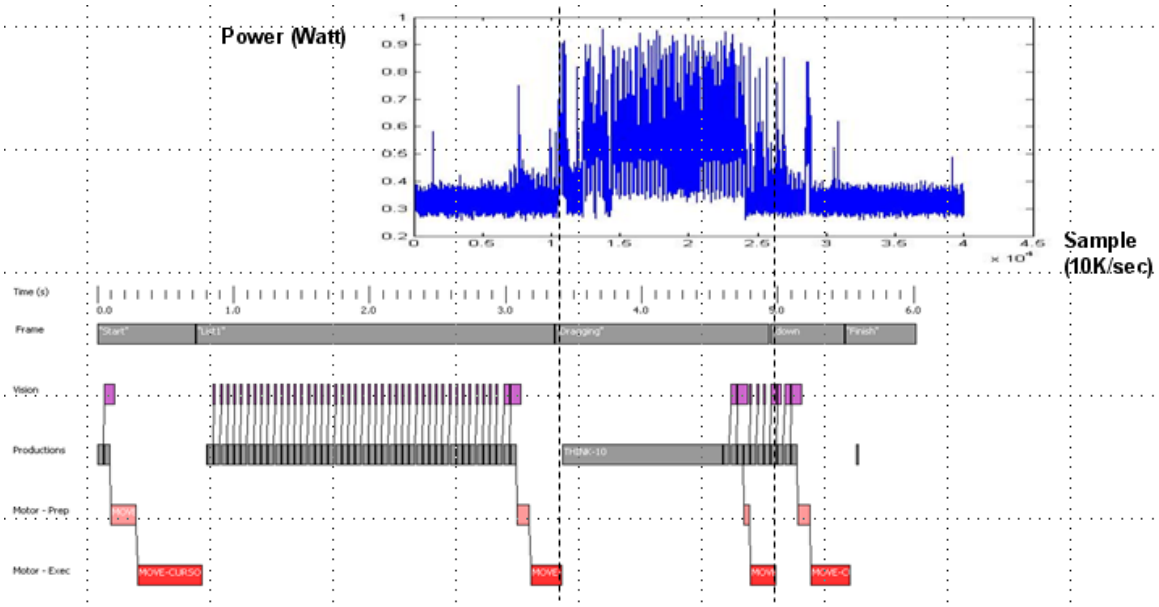


Figure 6.12: A "poor" mapping of energy profile and model visualization

Figure 6.14 shows four screenshots of the CWG application interface on the two platforms used. For simplicity, the storyboard is built for CogTool using the screenshots since the software is ready. In reality where KLEM is used during design time, modelers can use sketches of the proposed interface design to build the storyboard.

All tasks modeled have the same goal of finding the opening hours of the Metropolitan Museum of Art (MET). There are two different ways in CWG to find this information and each can

	iPaq	Tungsten
Vendor	HP	PalmOne
Model	Rx1955	T5
CPU	300MHz Samsung SC32442	416MHz Intel XScale
Storage	32 MB built-in RAM, 64 MB Flash ROM	215MB storage capacity, 160MB internal flash drive
Display	TFT color LCD, 64K colors, 240 x 320 (QVGA)	TFT color display, 64K colors, 320 x 480
OS	Windows Mobile 5.0	Palm OS v5.4

Figure 6.13: Specifications of target platforms

be considered as a different interface design. One way is to navigate the map of Manhattan area as depicted in Figure 6.14(c). The user can enter a more detailed map by tapping one of the four boxes representing different areas. When the street map of the MET neighborhood is displayed, the user can view the open hour information by tapping the dot representing the location of MET on the street map. The other way is to display a list of all New York museums, and choose MET from this list to display the open hour information. There are various methods to select MET from the list: searching or browsing. For the iPaq, the user can tap the letter "M" on the software keyboard at the bottom of the museum list as shown in Figure 6.14(b); tap the trough of the list's scrollbar until MET can be viewed in the list; tap the down arrow; tap the down arrow and hold it until the MET item appears in the current list window; drag the scrollbar; and press the hardware navigation button at the bottom of the device to browse down the list.

For the Tungsten device, the user can input the letter "M" on the software keyboard; tap the down arrow; gesture "M" in the Graffiti area at the lower part of the device display; and press the hardware button to browse down the list.

The scrollbar in the Tungsten does not have the same design as the iPaq and cannot be manipulated using dragging or tapping the trough. Although one can invoke the soft input panel (SIP) at the bottom of the iPaq screen, the handwriting area will obstruct the lower part of the list, which makes it unnatural and error-prone to use. Therefore handwriting recognition in iPaq that corresponds to the Palm Graffiti input is not used in these tasks.

To verify the KLEM prediction of user time and system energy, a user study is performed on another 10 participants (six male, four female), all are engineering majored undergrad or graduate students who are familiar and comfortable with using computers. Each participant is first asked to practice all the tasks under the author's instruction in a training session. The participants are given adequate time to practice until s/he became very familiar with the tasks without making errors or unnecessary pauses during task execution. The participants were then asked to perform all 12 tasks on the two devices during the testing session. The device power supply traces with corresponding time stamps of each task were measured during the testing session, as described in the previous section.

The average measured user times versus the predicted model times is shown in Figure 6.15. The time prediction errors against the average measured user time for the iPaq tasks are between 0.1% and 11.7% (average 5.6%). For the Tungsten, the error rates are 2.6% to 12.6% (average

(a) iPaq: Map Navigation interface

(b) iPaq: Scroll list interface

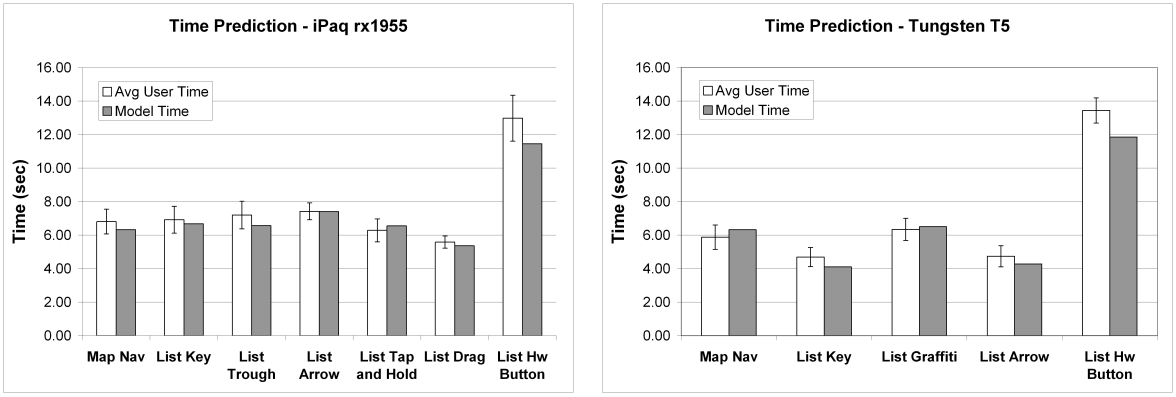
(c) Tungsten: Map Navigation interface

(d) Tungsten: Scroll List interface

Figure 6.14: Screenshots of CWG

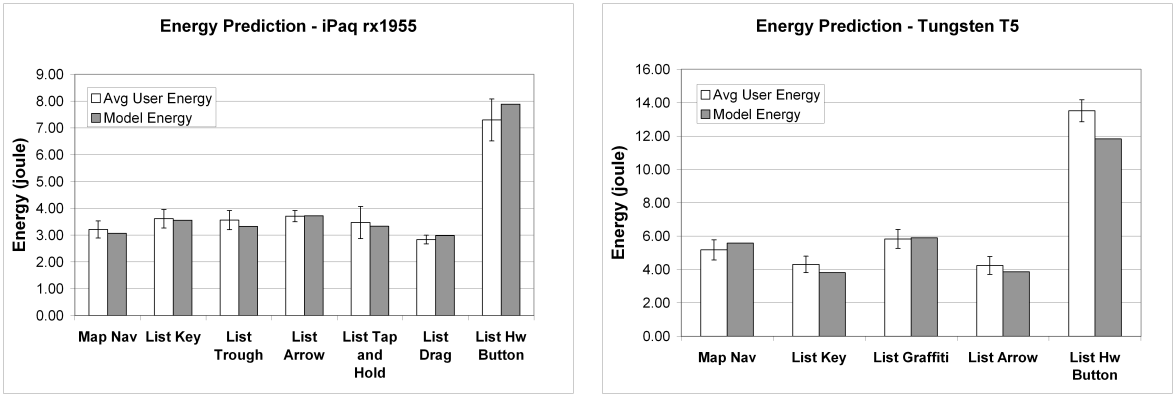
8.8%) for KLEM time predictions. Note that the predictions for “List Hardware Button" tasks for both platforms have comparably higher error due to the fact that the number of hardware button presses used by different users to browse the list varies widely.

The average measured task energy versus the predicted model energy is shown in Figure 6.16. The energy prediction errors against the average measured task energy for the iPaq tasks are between 0.3% and 8.1% (average 4.4%). For the Tungsten, the error rates are between 1.2% and 12.5% (average 8.4%).



(a) iPaq (b) Tungsten

Figure 6.15: Comparison of measured user time and model predicted time



(a) iPaq (b) Tungsten

Figure 6.16: Comparison of measured task energy and model predicted energy

6.3 Comparing design alternatives using KLEM

The previous sections presents the Keystroke Level Energy Model to predict the energy consumption of 12 tasks to obtain the same information on two mobile platforms. On each platform, the tasks were performed on a GUI based mobile interface equipped with various input modalities. Two different interface designs are studied, one the Map Navigation design, and the other is the List Browse design. For the latter, there are four to six different interface manipulation methods, or modes to find the same information on iPaq and Tungsten, respectively.

The terms modality and mode of user interfaces are often used interchangeably, but the impact of these aspects on both the user and the system can be significant, especially for mobile systems. A **modality** is a path of communication employed by the user interface to carry input and output. Examples of modalities in mobile computing include: (Input) stylus allows the

user to make selections or create drawing; (Output) screen allows the system to display text and graphics (vision modality) and speaker allows the system to produce sound (auditory modality). A **mode** is a distinct method of operation with a computer program, in which the same input can produce different perceived results.

To achieve the same goal, there are usually various user interface modalities and modes for the designer to choose from. Different design decisions can make significant difference in important system aspects such as energy consumption. Such design alternatives also include the selection of software and hardware platform, as well as operating systems, which is beyond the scope of this dissertation. The results in Section 6.2 observed up to a factor of three variation in doing the same task with the same interface design on different platforms.

The accuracy of model prediction of total user time and task energy of the two design alternatives was compared in the previous section. From Figure 6.15 and Figure 6.16 it can be seen that the time and energy for doing the same task using different modes on the same platform (List Hardware Button vs. List Drag or List Key) vary by a factor of two to three.

Three set of examples are chosen including the tasks defined in Section 6.2 to demonstrate the difference in design alternatives.

Example 1: Using different modes The energy profiles of the last five methods (modes) for list browsing on iPaq in Figure 6.16 are shown in Figure 6.17. The List Trough mode only contains two taps and two display updates. However, in the other four modes, all methods used to manipulate the scrollbar to browse the list require trigger frequent UI events and display updates. The overhead for processing UI events for the "List Hw Button" mode is extremely high, thus leads to the highest task energy in all methods.

Despite the difference in the power profiles of these modes, the difference in task energy consumption is not as significant. This is because the total task execution time for all modes is less than 9 seconds, and most of the task energy is spent on loading the list and searching the correct item, while the list manipulation only consumes a negligible amount of energy. In cases where there are large amount of manipulation operations in the task, the choice of manipulation modes can play a very important role in the total task energy consumption.

In addition, from an energy optimization point of view, there is much longer user idle time between operations in the "List Trough" mode, which could be utilized by the application or OS to apply energy reducing algorithms. While the other four modes may provide little opportunity for energy optimization due to the frequent need to process interrupts from the UI.

Example 2: Using different modalities All tasks discussed in last section are based on a GUI modality using regular input devices – stylus and touch screen. A design alternative that uses non-GUI modalities can be a speech-based interface.

Because off-the-shelf applications are used in this study, there is currently no speech interfacing capability in the CWG application. Another off-the-shelf software called MobileSpeech for the iPaq platform is used. MobileSpeech can recognize natural language sentences such as “Look up contact John Smith”, then searches John Smith’s information in the local database, and dis-

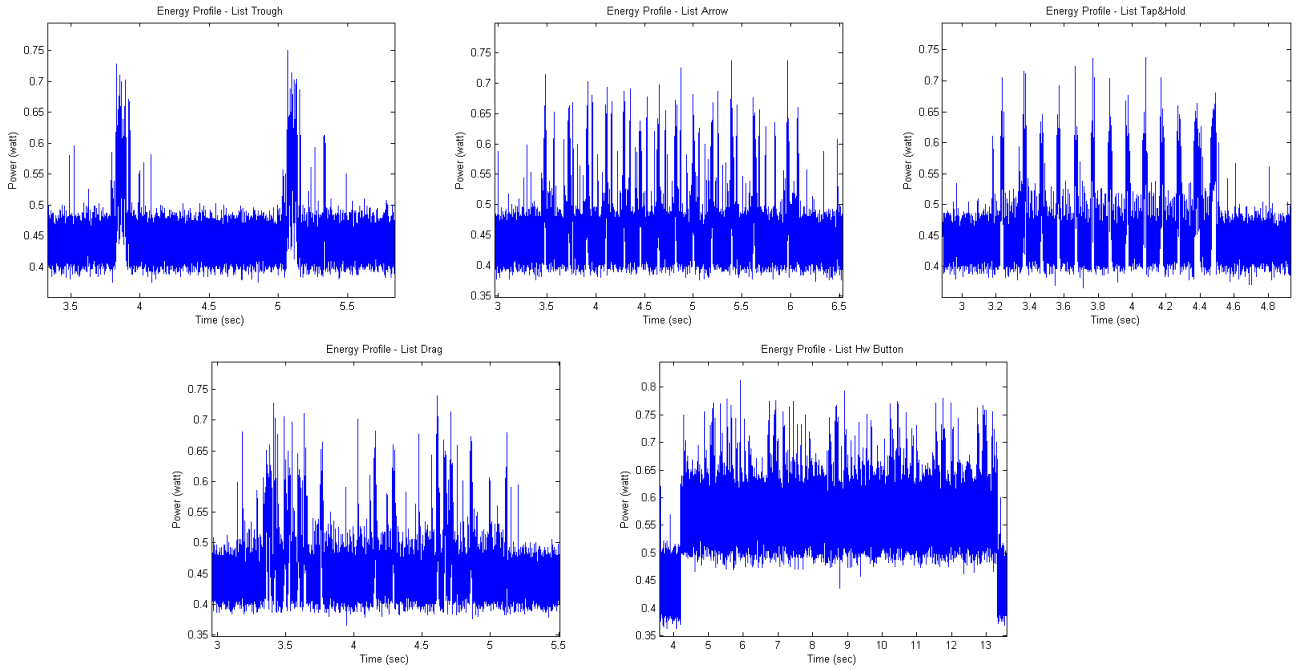


Figure 6.17: Energy profiles of list browsing methods

plays this information. This is enough to simulate the information query functionality of CWG. MobileSpeech is used to recognize the sentence "Look up information MET", and to display the information of the museum. The power profile in Figure 6.8 is obtained from MobileSpeech.

Figure 6.18 compares the total task time and energy consumption when using speech (*Speech no LCD & Speech*) modality and GUI (*List Drag*, *Map Nav*, *List Tap and Hold*, *List Trough*, *List Key*, *List Arrow*, and *List HW Button*) modality. The tasks are listed and sorted by energy consumption (low to high). The execution time of each task is also shown in Figure 6.18. Although the Speech task ranks the second highest in energy consumption (about 20% higher than the List Arrow task), it can achieve 60% energy savings over the List HW Button task. Because the device display can be turned off during speech-based interactions, the energy consumption of the Speech no LCD task is the lowest of all – about 32% lower than the List Drag task and by a factor of three to the List HW Button task.

Example 3: Using different implementation Another set of design alternatives comes from the input methodologies of handheld devices. In general, the input methods for mobile devices can be divided into three categories: Letter Recognition, Transcript Recognition, and Soft Keyboard. Letter Recognition allows the user to draw one letter at a time, recognizes it into text, it then accepts for the next drawing from the user. Transcript Recognition, on the other hand, allows the user to handwrite in a more natural way as he/she would write on paper. The user pauses after writing a word or the entire sentence, and this is when the device recognizes the handwriting into text. Soft Keyboard is very straightforward: the user taps the letters on the

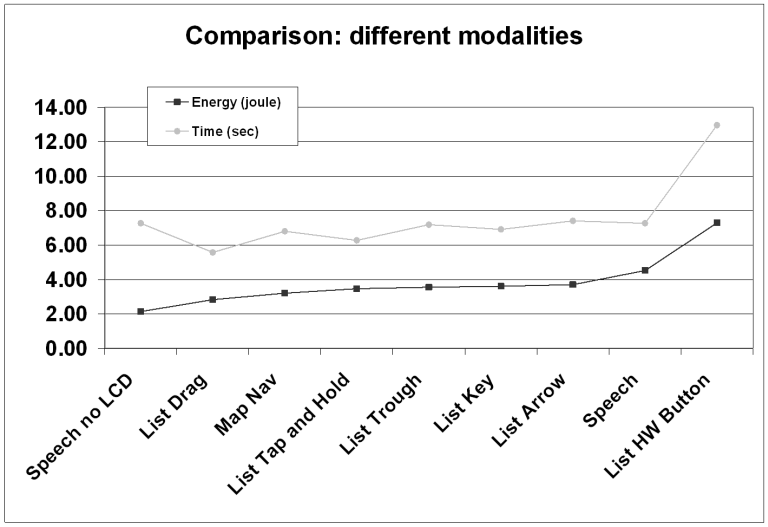


Figure 6.18: A comparison of energy and time using different interaction modalities

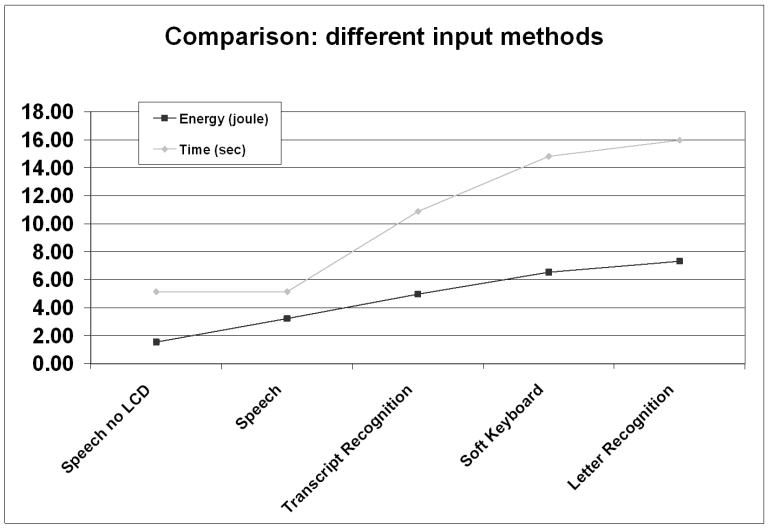


Figure 6.19: A comparison of energy and time using different input methods

small on-screen keyboard, and the device displays the text.

The time and energy of the three text input methods are compared with a speech-to-text application and the comparison is depicted in Figure 6.19. The power profile for speech preprocessing and decoding of the synthetic speech-to-text application is also obtained from benchmark measurements using MobileSpeech. The speaking speed of the user is set to 105 words per minute because people tend to dictate to computers at this speed [6]. In this comparison both speech-base input methods (with or without display) have the lowest time and task energy. This is because the speed of speaking is much faster than writing, say nothing of pick up single letters using

one stylus on the small soft keyboard. Despite the obvious advantage in speaking speed, speech recognition is equivalent to Transcript and Letter Recognition interfaces for processing overhead of text recognition.

6.4 Summary

This chapter presented the detailed process of the Keystroke Level Energy Modeling methodology that extends the Keystroke Level Model. KLEM can be used to quantitatively predict both the user performance and energy consumption of interactive tasks performed on mobile systems. KLEM assumes skilled user performing error free, serial interactions with mobile interfaces, and models single application without other concurrent processes in the system. A set of benchmarks were designed to obtain the necessary interfacing and energy profiles of the system under study. KLEM can predict user time and energy consumption from story boards of proposed user interactions with good accuracy. It also serves the designers of mobile systems as a convenient tool to compare and make early decisions among different design options, and to resolve potential design issues before investing in actual user testing and iterative development. While KLEM was presented with a focus on handheld devices, the methodology can be applied on any mobile interactive system. KLEM can be used to address the limitations identified in these two initial efforts, and can help user interface and interaction designers to profile and understand the user performance and energy consumption at an early stage of system design.

Chapter 7

Related Work

This dissertation advocates on integrating energy efficiency as an important metric of user interaction design in mobile systems. Such a design should aim to minimize the energy consumption of a task as well as to increase user performance/productivity and satisfaction. To the best of our knowledge, this dissertation is one of the first efforts that consider energy as a first-class usability metric in mobile systems.

While there is no single research effort that spans all the subjects discussed in this dissertation, there is a large amount of work that intersects one or more areas. This chapter discusses work related to each part of this dissertation. Section 7.1 summarizes related work on improving the energy efficiency with focuses at the software level. Section 7.2 surveys existing approaches on energy characterization, modeling and simulation. Finally, Section 7.3 discusses more ad-hoc research efforts on improving the energy efficiency of mobile user interfaces. This section also discusses related approaches on characterizing and improving mobile interaction performance.

7.1 Energy optimization

The optimization of energy consumption can be done at different levels of the computer system architecture [75]. Hardware components are the immediate consumers of energy, and the major hardware components in a mobile system include the processor, memory, secondary storage (e.g. flash memory or hard disks), network interface, display, and other interfacing hardware. Hardware energy consumption can be optimized at the circuit level using techniques such as clock gating, supply voltage scaling, and supply voltage gating to reduce both dynamic and leakage power [27, 28, 170]. At the architectural level, energy optimization techniques typically detect idleness of components and appropriately transit them to a lower power consuming state [23]. However, hardware level power and energy optimization is beyond the scope of this dissertation, hence will be discussed no more in this chapter.

This section surveys techniques that optimize energy consumption at the software level – the operating system (OS), compiler, and the application. It is a consensus that the potential for energy savings in software is greater than the potential for savings in hardware, but that the

software savings are more difficult to achieve [86]. Therefore, there is a large body of existing work at the lower levels of the software, i.e. at the operating systems and compilation levels. At the application level, most efforts have been trying to gain an understanding of how application software affects energy consumption.

In general, software energy optimization techniques take one or a combination of three major approaches that will be discussed in the following three subsections.

7.1.1 Activity adjustment

Techniques in this category intuitively reduce energy consumption by reducing the activities of software that consequently causes hardware activities. As discussed in Chapter 2, on the one hand, if one application or algorithm takes shorter time to execute than another, then the system will consume less energy. On the other hand, if one application or algorithm uses hardware resources more smartly than another, those resources can be put into low power modes to save energy. Example activity adjustment approaches include improving memory access locality and reducing unnecessary network accesses.

Most compiler optimizations that aim at reducing application energy usage fall into this category. Tiwari et al. [155, 156] modify the compiler to take into account the power consumption as well as the timing cost of each instruction, and examine instruction-level energy optimizations such as instruction reordering and energy-driven code generation. The major energy savings come from reducing the time to complete a computation, not from using lower power instructions. Simunic et al. [145] examine energy optimizations for a MPEG application on a StrongARM processor and note that compiler optimizations only produce a 1% energy benefit for the application, while hand-crafted source code optimizations produce a 35% energy benefit. However, it is not clear how much of the energy reduction is due to the shorter execution time because a 32% reduction in execution time is reported for the hand-coded implementation. Algorithmic transformations in applications have been shown to give significant power savings [147].

At the application level, energy optimization can be achieved by balancing the quality of service (QoS) of the software. An example of such quality is data fidelity. Flinn et al. introduce energy-aware adaptation [50, 52, 113] that extend the Odyssey platform to adapt application specific data fidelity (e.g. video quality for a video player, and vocabulary size of a speech recognizer) for multiple applications based on predicted energy demand and targeted battery lifetime. Shenoy et al. [137] propose to transform the requested network data stream to reduce receiving and decoding energy. Mohapatra et al. [107] also investigate energy optimizations from adapting the quality of streaming video. Fei et al. [46, 174] propose and implement a user-level coordination framework to adapt multiple applications for energy savings. The framework makes trade-off selections between energy conservation and application QoS.

In addition to reducing activities, energy optimization techniques in this category also balance the activities among different hardware components to reduce the overall energy consumption. Barr et al. [16] find that overall energy consumption can be reduced by compressing data before transmitting them through the network interface. Anand et al. [11] propose to determine whether

to retrieve files from the local hard drive or through the network interface based on the power-saving modes they are in. On the other hand, computation offloading and remote execution techniques [97, 116, 129] outsource compute-intensive tasks from a mobile system to a wall-powered computer through a network interface.

7.1.2 Mode switching

Mode switching techniques put a hardware component into a different power mode when necessary. Hardware components provide power-saving mechanisms that enable software energy management through standard interfaces. Mode switching strategies require knowledge about different power modes of a hardware component, as well as the information about its current and future activities. Predicting the future functionality requirements have been the most challenging topic for mode switching strategies.

Most OS level energy optimization techniques fall into this category. Because the OS has a view of the overall state of the system, it is in a better position to judge whether a device should be put into a low power mode than the device itself. The OS is also in a better position to judge than an application because it can balance the needs of several applications, without the need of modifying the applications. The Advanced Power Management (APM) specification [5] allows the operating system to query the power state of devices such as the hard drive and place these devices into low-power modes. The Advanced Configuration and Power Interface (ACPI) specification [1] is intended as the successor to APM, and allows detailed power management of individual hardware components. The Milly Watt project [44, 158] explore the development of a power-based API that allows a partnership between applications and the operating system in setting energy use policy. They present a power-aware page allocation policy coupled with dynamic hardware policies that can dramatically improve memory energy-efficiency [93].

For reducing the transition latency, Lorch et al. [99] consider strategies for a number of different subsystems, especially within the constraints of the design philosophy of MacOS. Li et al. [95] perform a quantitative analysis of the potential costs and benefits of spinning down the disk drive as a power reduction technique. Paleologo et al. [118] consider the overhead requirements of making a transition from one power management state to another and introduce a stochastic model for evaluating power management policies.

Besides the OS, the compiler can generate code and data transformations to increase idleness of hardware components so that they can be transitioned to low power modes more efficiently [39]. Lu et al. [103] use user-level power managers to collect utilization information from hardware devices and control the power state of the devices. This work is later extended by Simunic et al. [143] with a specific model of hardware device usage patterns based on time-independent semi-Markov decision processes. They show that their approach can achieve better results than alternative policies when managing processor, hard disk, and wireless network power states.

7.1.3 Using alternatives

Techniques in this category use secondary, lower power components to house-keep the basic functionality of the primary component, which can then be turned off or put to idle mode whenever possible. For example, flash memory can be used as a low-power alternative to magnetic disk or disk cache. For wireless interfaces, low-power listening device can be used to check beacons during the interface's sleep mode.

Douglis et al. [42] study the possibility of using flash memory as a replacement for hard drives and report low energy consumption, good read performance, and acceptable write performance. March et al. [106] examine using flash as a second-level buffer cache to reduce energy by allowing the hard disk to be idle more often. Schlosser et al. [134] note that MEMS-based storage may have significantly less power requirements than traditional storage devices. Shih et al. [138] propose to greatly reduce the idle power of a wireless LAN enabled PDA phone by turning the phone off and using a secondary, low-power wake-up mechanism. Part of the research presented in Chapter 3 takes a similar *using alternatives* approach that suggests using other low-power, secondary user interface devices, such as an Light Emitting Diode (LED) email indicator, to allow the main display module to stay longer in the off mode.

7.2 Energy characterization

Another body of work related to this dissertation analyzes and characterizes how energy is consumed in mobile computer systems. Some of these efforts characterize energy consumption of one individual hardware component of the system, some break down system energy usage onto different hardware components, and some specifically characterize the energy consumption of operating system, application software, or user interface.

This Section discusses two major approaches taken by these efforts: the measurement based approach and the analysis based approach. Measurement based approaches are based on actual hardware and software implementations and can potentially produce more accurate results, but the costs for benchmarking and measuring can be high, especially for very complex systems. Analysis based approaches usually use a model of the energy cost and estimate the energy cost using simulations, and hence do not require physical hardware implementations, but the detailed nature of the model limits its scalability to large, dynamic systems. Although a larger portion of existing work takes the simulation based direction, the distinction between these two approaches are very small and many energy characterization efforts employ both.

7.2.1 Measurement based approach

Measurement based approaches first profile the energy consumption of specific system activities. The measurements are used later, during execution, to calculate the total energy usage by accounting for the number of times each activity occurs.

Lorch et al. [98, 100] develop PowerMeasure and StateProfiler that provide energy measurements for Apple Macintosh laptops. PowerMeasure benchmarks the various power states of hardware components such as the processor and the hard disk. StateProfiler records the transitions between power states and uses the benchmark data to estimate total energy consumption during application execution. They also show how power-saving features affect the breakdown of overall power consumption so that the success of new software techniques and hardware changes at reducing power consumption can be estimated.

Flinn et al. [51] present PowerScope, a tool that profiles energy usage of applications by mapping energy consumption to program structure. Their approach combines hardware instrumentation to measure current level with kernel software support to perform statistical sampling of system activity and employs post-processing to map the sample data to program structure and produces a profile of energy usage by process and procedure.

Farkas et al. [45, 49] measure the energy consumption of the Itsy pocket computer and quantify the energy costs of Java design trade-offs. They use a number of micro-benchmarks to measured the power and energy consumption of various hardware components. Their measurements show a wider range of dynamic power demand of the Itsy than does the ThinkPad computer.

On individual hardware components, Stemm et al. [151] were the first to present a power characterization of several wireless network interfaces on handheld devices and point out that power management of the idle time is much more important than reducing data transfer for energy efficiency. Raghunathan et al. [123] present an power analysis of a multi-radio mobile platform. Their analysis focuses on contrasting the power efficiency of various wireless technologies and highlights the trade-offs between the computation, storage, and communication subsystems, but does not include a display subsystem. Zedlewski et al. [171] present Dempsey, a simulation environment that characterizes disk power consumption use stimulus-based measurements to extract power parameters without using detailed manufacturer specifications.

Most OS level energy characterization approaches have a focus on embedded operating systems because the overall energy consumption depends very much on which OS is used and how the OS is used in such systems. For example, Acquaviva et al. [10] analyze the energy overhead due to the presence of an embedded OS in a wearable device.

7.2.2 Analysis based approach

For modeling processor energy consumption, power analysis can be done at the instruction level or architectural level. Both are targeted at exploring alternative hardware architectures and compiler optimizations.

Instruction-level power analysis is first introduced by Tiwari et al. [152, 156]. They construct per-instruction energy models for several processors with energy costs on inter-instruction switching. Their method is based on the hypothesis that by measuring the current drawn by the processor as it repeatedly executes certain instructions or short instruction sequences, it is possible to obtain most of the information that is needed to evaluate the power cost of a program

for that processor. Klass et al. [89] develop a model that closely approximates inter-instruction energy effects, but requires much fewer measurements to construct.

Architecture-level power analysis uses architectural layouts to build detailed power models for each of the internal modules of the processor and estimate the power and energy consumed by program execution. Wattch [22] and SimplePower [160, 169] are two examples in this category: both approaches extend the SimpleScalar framework to provide power estimates. A similar approach is used by Simunic et al. [145] to develop a cycle-accurate simulator specialized for the SmartBadge platform.

There are many efforts that model the power and energy consumption of real-time operating systems (RTOS) [17, 40, 152, 153] and provide simulation frameworks to analyze the execution behavior and energy consumption of RTOS's. Gurumurthi et al. [63] present SoftWatt, a complete system power simulator that models the CPU, memory hierarchy and a low-power disk subsystem and quantifies the power behavior of both the application and operating system.

Cignetti et al. [32] create an energy model for the PalmOS™ family of mobile devices. Based on the device's hardware subsystems, their model identifies a set of discrete device power states and transitions between the states and is used to extend the Palm OS Emulator (POSE) into an energy simulation environment. Zeng et al. [172] propose the Currentcy Model that unifies energy accounting over diverse hardware components and enables fair allocation of available energy among applications. They implement the ECOSystem that treats energy as a first-class operating system resource and supports explicit control over the battery resource, in order to extend battery lifetime by limiting the average discharge rate and to share this limited resource among competing tasks according to user preferences.

At the software level, power and energy estimation and optimization techniques for embedded software have been investigated extensively [109, 144, 154]. Sinha et al. [146] present JouleTrack, an web based tool and methodology that estimates the energy cost of embedded software running on two microprocessors. Xue et al. [168] propose an object-oriented energy model for embedded software, which also provides interfaces to the energy models of underlying hardware components.

Seo et al. [136] define a framework that employs a computational energy cost model for software components of Java-based embedded systems, both prior to and during run time. Xian et al. [165] present an approach to assign energy responsibility to individual processes based on their impact on power management and to estimate potential energy reduction by adjusting the processes. They then present a programming environment that allows energy-aware programs to identify different ways to achieve the desired functionality and choose the most energy-efficient option at run-time [166]. The energy estimation is based on run-time energy characterization that records a set of run-time conditions correlated with the energy consumption of the options.

7.3 Energy and mobile user interaction

The significant growth of mobile computing has brought more and more research interest on energy efficiency from the user interaction perspective. This section discusses related research

efforts that also take this perspective.

7.3.1 Human factors in energy optimization

Although many energy management and performance scaling efforts [47, 62, 101] use interactive applications as benchmarks, these efforts ignore the interaction between the human user and the system and treat it in the same fashion as compute-intensive tasks. Human-perceptual thresholds are used [47, 101] to guide system performance scaling and respond before the user can perceive the delay, but no information about the real workload and context of the user is applied. Lorch and Smith [102] find that user interface events incur different computation loads and propose to conduct performance scaling based on user interface event information during periods of busy system activity. By contrast, Zhong and Jha [174] use user interface information to predict system idle time for energy management. Compared with these existing approaches, this dissertation emphasizes the importance of efficient user interaction in optimizing system energy consumption, and integrates energy efficiency into user interface designs.

Dalton et al. [38] propose to use sensors and cameras to detect user presence for power management and point to a new direction for user- and context- aware energy management. The drawback of this approach is the high energy overhead caused by the detection method. By comparison, the approach presented in Chapter 3 detects the screen area where the user is focusing on and reduces unecessary energy expense on other areas.

7.3.2 User interface energy optimization

Chapter 3 has presented energy optimizations for the display subsystem of mobile user interfaces by adapting the output of the displays. Most display energy optimization efforts are for LCDs. Flinn et al. [50] evaluate the energy benefit from reduced computation with lower fidelity of images and video. They also propose a zoned back-lighting method to allow independent control of illumination level for different regions of the screen, although no existing display supports such zoned back-lighting yet. Other studies also evaluate scaling the back-light with corresponding changes to compensate for any fidelity losses [30, 56, 139]. Kamijoh et al. [84] study the energy trade-offs for the IBM wristwatch computer and discuss the energy impact of controlling the number of illuminated pixels and reducing the brightness of the screen (e.g., at night). Choi et al. [31] perform a detailed power characterization of the display subsystem of a handheld device and propose optimizations that utilize varying fresh rate, color depth, and back-light luminance. Cheng et al. [29] exploit limitations in human visual perception to reduce the power consumption of traditional LCDs and present an algorithm for minimizing power consumption of back-lights in color sequential displays. Embracing the emerging OLED display technology, Xiong et al. [167] design a micro-controller based driver for OLEDs.

In contrast to these efforts, Chapter 3 primarily focuses on the content and intent of the display output. The detailed characterization of display usage patterns is used to identify and understand the common mismatches between workload/user needs and current display proper-

ties. Chapter 3 also explores several new user interface and hardware designs that allow energy-adaptive control on the portions of the screen that are not of immediate relevance to the user, while continuing to provide complete functionality on portions of the screen of relevance to the users.

Zhong et al. [173] are the first to extensively study the energy efficiency of graphical user interfaces and I/O mechanisms in handheld devices. Their work focuses on the power the display subsystem consumes in the creation and manipulation of the images in framebuffers and identifies the relative energy trade-offs with various GUI event-handling and window-creation functions and their sensitivity to size, color depth, color sequence, and platform. They then suggest GUI designs that accelerate user interaction, minimize screen changes, minimize text input, reduce redundancy, and allow intelligent overlap of computation to reduce energy [175]. They also discuss the sensory perception-based limits for visual and auditory output, and I/O speed for various interfaces and propose a low-power interface cache that can enable energy savings during interactive tasks. Vallerio et al. [159] study the effect of user-interface design on energy efficiency and propose several GUI optimization designs to reduce the average system energy consumption of three benchmark tasks without sacrificing application performance. In comparison with Chapter 5 and 6, their work does not answer the questions of the interactions between the processes of human perception, cognition, and motor speeds, and GUI components.

Some of the latest research efforts bring forth the notion of "human-battery interaction" (HBI) of mobile systems. Froehlich et al. [55] present a small-scale battery study on four participants for two weeks as a case study of their in situ survey tool MyExperience and find that significant battery capacity remains on the device upon recharge and less than 30% of the recharges are driven by "low battery". Banerjee et al. [14] also conduct a user study on battery use and recharge behaviors on laptop computers and mobile phones and also find that users frequently have excess energy remaining in their batteries at recharge. They present an experimental system that adaptively adjusts the quality of service to meet the estimated battery requirements. Rahmati et al. [124] report a complementary study of user-battery interaction on mobile phones with a focus on how mobile users decide to recharge and on applying this knowledge to user-adaptive system energy management.

7.3.3 Performance of mobile user interaction

There have been several efforts to improve the performance of mobile user interaction. Since this dissertation mainly studies mobile user's performance using cognitive models, this subsection discusses related work on user characterization. In the user studies conducted in Chapter 5 and 6 the users perform tasks while sitting. For mobile user interfaces, the evaluation techniques would ideally take place in their natural settings. Consolvo et al. [35] use the experience sampling method that researchers in psychology have found to be effective for learning about situations and person-situation interactions. The technique compares most closely with recall-based self-reporting techniques such as interviews, traditional surveys, and diaries. Intille et al. [74] describe three tools for acquiring data about people, their behavior, and their use of technol-

ogy in natural settings. These tools can provide researchers with a flexible toolkit for collecting data on activity in homes and workplaces, particularly when used in combination.

To characterize data entry performance on mobile interfaces, Dunlop et al. [43] use a KLM to evaluate their predictive text input method which works almost identically to T9 Text Input® and Multi-tap. James et al. [77] conduct a study to obtain performance data for entering text on a mobile phone in order to compare it to performance predictions based on two different mathematical models. Speed data was obtained for two text input methods, T9 Text Input® and Multi-tap. Pavlovych et al. [119] present a model for predicting text entry speed on a 12-button mobile phone keypad. Cox et al. [37] show that the most efficient input method involved combining key-press navigation with spoken text-entry. However their user study participants show a stronger preference for uni-modal spoken input, despite a decrease in the efficiency of interaction. They also find speech more error-prone than current predictive text entry technology. Cao et al. [24] present a quantitative human performance model for making single-stroke pen gestures within certain error constraints in terms of production time.

7.3.4 Applications of KLM

Initially, the KLM was targeted mainly at text editing tasks on office desktop computers [25, 26]. Although it sometimes seen as a drawback of such models to assume error-free, expert user interaction, the KLM has since then revealed remarkably precise prediction results in various applications such as manual map digitizing [67], vehicle navigation systems [105], and email message organization [13]. In some cases where experimental studies indicated that estimates in fact were considerably off the actual values, the estimated difference between two examined designs still proved to be a strong basis for making a choice between them [43, 91, 111]. Pettitt et al. [120] propose an extended KLM to predict measures based on the occlusion protocol of in-vehicle information systems and conclude the considerable merit of the extended KLM as a first-pass design tool.

The initial work in [43] is improved by How et al. [72], where the presented model is more fine-grained. They define 13 operators that map onto the phone keyboard interface more closely with different input methods and gather new times from video taped sessions. Pavlovych et al. [119] further consider novice users using the cognitive load operator to model input verification. Although their models give only rough approximations to real user behavior for text input, they can still correctly predict which input methods are faster than others (e.g., predictive over multi-tap). There is also work reporting on time measurements for key presses and the mental act operator for text input in different languages (e.g. Myung for Korean [111]).

In addition to text input, Mori et al. [108] study how the time values of the original KLM operators apply to mobile phone menu navigation and conclude that the operator values fit quite well and suggest only minor modifications.

More and more modeling efforts have recently focused on mobile interaction. Holleis et al. [70] extend KLM for advanced interactions with mobile phones targeted at pervasive services, including near field communication as well as built-in cameras and sensors. In another work,

Holleis et al. [69] integrate KLM in the design of tangible interfaces and present two ways of using KLM: through a specifically designed application for prototyping in-car interfaces and through a rapid prototyping environment for general tangible user interfaces.

Chapter 8

Conclusion

8.1 The prospect of mobility

Mobile computing has exploded to provide unprecedented new user experiences. Mobile devices have been used primarily as personal organizers and social connectors because of their immediate availability, despite their inherent limitations that lower productivity. Currently, the mobile web has just taken off as the promise of immediate access to information, entertainment, and commerce services. Simple and speedy access are critical to achieving this promise.

Research in mobility should endeavor to understand how to provide the easiest and most efficient user interaction experience to facilitate the access to service and enhance users' everyday lives. No user would be interested in taking complicated, lengthy steps to access information. It is also very important to focus on helping users to accomplish their goals in a timely manner. For instance, if the user is not sure which intersection to make a right turn in fast moving traffic, s/he needs to know it now, not five minutes later. In addition, the environments in which mobile systems are used are likely to have many distractions competing for users' attention. Therefore the contexts within which users use mobile devices must be thoroughly understood.

8.2 Contributions

This dissertation has taken a step to address the unique challenges and strengths of mobile computing, with a focus on energy efficient mobile interaction design. Energy efficiency should be considered as an important goal of user interaction design in mobile systems, because most tasks on mobile devices are interactive by nature, and battery life is one of the biggest concerns of all users.

This dissertation started with a detailed study of typical workload and screen usage on laptop and desktop workstations. Gaining the insight that users only use 50% of the screen size for their average workloads, Dark Windows, an energy efficient display design was presented and implemented. This design optimizes display power consumption by adjusting the color and illumination of different screen regions corresponding to the user's current workload.

As an early attempt on the energy efficiency of mobile user interface, there are limitations in this work. First, the user study was conducted on PC and laptop workstations, because of the low prevalence of mobile devices at that time. However, subsequent user studies by HP researchers including those who collaborated in this work [20, 66, 125] have shown solid proof of good user acceptance of the Dark Windows design, especially with its energy benefits on handheld PDAs. Secondly, due to the unavailability of OLED panels then, the energy benefit of this study was based on analysis of manufacturer data. It would be beneficial to obtain more data from measurement on comparable OLED and LCD displays.

To explore another dimension of the usage space of mobile interfaces, this dissertation next presented AstroRDS, a mobile computing system and network infrastructure that displays documents and information on user's mobile devices on ambient ubiquitous display resources, and facilitates much better viewing experience for the user with comparatively ease of use. The energy consumption and network usage of AstroRDS was tested in comparison with the a system that uses the VNC remote desktop protocol, and AstroRDS saves several orders of magnitudes on energy and bandwidth for viewing and controlling tasks, with similar initial loading overhead as VNC. However, the lack of a user study makes it difficult to decide the user experience of such ambient display system and the possible benefits or drawbacks. Some user studies on these aspects would be useful for future improvements.

The limitations identified in these two initial efforts revealed a challenge for designing mobile interface for energy efficiency. The lesson learned is that in order to understand the user experience and energy consumption, a good deal of effort needs to be spent on experimental setups, and even more on user studies. This motivates the presented initial exploration on using cognitive modeling techniques to produce quantitative a priori predictions of both the user performance and energy consumption. Therefore, this dissertation discussed the applicability of cognitive modeling, i.e., KLM, on mobile user interaction tasks, and addressed new modeling issues for mobile interfaces. Then, the dissertation presented KLEM, a quantitative analysis methodology that predicts both user performance and system energy consumption of an interactive task, and during early design phases. The KLEM extends the KLM, and integrates system energy consumption with the user interaction model of the KLM. A set of benchmarks were designed to obtain the necessary interfacing and energy profiles of the system under study. KLEM can predict user time and energy consumption from story boards of proposed user interactions with good accuracy. It also serves the designers of mobile systems as a convenient tool to compare and make early decisions among different design options, and to resolve potential design issues before investing in actual user testing and iterative development. While KLEM was presented with a focus on handheld devices, the methodology can be applied on any mobile interactive system.

In summary, this dissertation makes the following major contributions:

- It has characterized the impact of hardware, software, and human user on the energy consumption of mobile systems, and pointed out the significance of user interaction design for energy efficiency.
- It has presented Dark Windows, a user interface design that optimizes energy consumption

of light emitting displays by adjusting screen content according to the user's workload and focus.

- It has described the Astro remote display system that facilitates utilizing of ambient ubiquitous display resources and optimizes user experience and energy efficiency when viewing the information on their mobile devices.
- It has verified the Keystroke-Level Model on predicting user performance and introduced new operators that are unique to mobile user interfaces.
- It has brought forth the Keystroke-Level Energy Model, a quantitative methodology that extends KLM and predicts both user performance and system energy consumption of an interactive task during early phases of design.

8.3 Future work

The work presented in this dissertation can be extended in several ways and also applied to other areas of research on mobility and energy.

Future research on energy efficient user interface subsystems described in Chapter 3 needs to identify and refine the interface design principles that support reduced display power consumption and offer a positive or enhanced user experience. User study shows the desire to use contrast to highlight areas of interest, personalize an interface, and view a large amount of context. For example, interface personalization offers a fertile research area, including an assessment of pre-existing display settings so that each settings provides a unique combination of energy consumption and interface color and illumination. Also, interfaces that involve controlling a temporal aspect offer much promise. For example, recently changed areas of the screen could be displayed brightly, then fade as time progresses. Finally, additional work must be done to determine the amount and nature of power control the system should expose to the end user.

Chapter 4 showed the core functionality and features of remote display and control. The ultimate goal however is to allow any displayable data type to be used within this system. Furthermore, the system would allow much smaller and more interface-limited devices to utilize the system's functionality. The current system utilizes a standard size PDA as the mobile device. A fully mobile user would be served better by even smaller devices, possibly worn on the body, that do not require significant user attention to perform their function (e.g. diverting attention to a screen, both-hands use). For data types such as text documents or graphics like maps that are mostly viewed with simple controls, such as page-up, page-down, zoom and scroll, the mobile device only needs to have button inputs to trigger these actions without a feedback interfacing component, such as a display. Envision one such device as the size of a key chain fob, similar to today's key chain USB memory keys. The device would have a few indicators for power level, on-off state, and connection state. An SD card slot or other standardized, miniaturized flash memory connector would be sufficient to provide storage for the user's data. A low-power, short range radio provides the interface to the ambient resources. The controls would consist of a small 8-way rocker switch and a few buttons, that work together to provide page and screen

navigation capability. When connected to an ambient display, a menu on-screen would show all the available files in the mobile device's storage and allow display and navigation of a file's data. Since the key chain device does not have extensive screen components, the issue of selecting a particular ambient display in an environment where many are present is more complicated than in the case of our prototype system. One idea is to employ an infrared beacon (or another type of limited range light signaling) on the ambient display and a passive photo-detector on the key chain device. When the key chain device is brought near the ambient display it detects and decodes the beacon pulses, which contain enough "bootstrap" information to allow the key chain to make a connection via its radio link.

In addition to providing upload-and-view functionality, the ambient displays that also have network connections to the outside world can act as email and other information proxies, collecting this data on behalf of the user and transferring it to his key chain device when a connection is made. The data could be viewed immediately during that session, or stored on the key chain device's storage card for viewing later, either on another ambient display, or a conventional computer.

Another extension would be seamless session transfer when mobile devices utilize ambient computing and display resources. Based on user preferences, session mobility should be able to transfer an on-going task session from a mobile device to an ambient display resource, as well as among devices. Examples of such a session include a web browsing session with page contents, cookies, browsing history and bookmarks, a multimedia session with viewing preferences, quality and resolution, and resuming point. The session handover should be fast and require very little effort from the user. It should also take into account the difference in capabilities of devices.

As an initial attempt, KLEM currently addresses serial user interaction, hence tasks being modeled need to be represented in a sequence of primitive operations. The tasks used for verifying KLEM are comparably simple, sequential, computing only tasks. In reality, mobile tasks could involve more complex user activities rather than serial operations, including situations where multi-modal, multi-tasking, and distractions occur. Future extension of the model would need to address these situations. It would also be useful if the KLM part can be refined to address mobility, since most of the original and mobile KLM operators are based on studies where the users are stationary. In reality, users send text messages on the bus, browse the web when running on treadmills, and more and more interactions happen on the move. The modeling and designing should take into account the context and mobility challenges.

KLEM currently provides only characterizations and predictions on user performance and system energy, which can be used to compare design alternatives. In the future, it would beneficial if energy and usability optimizations can be derived from the KLEM predictions. One possible optimization would be using the mental preparation durations as predictors for system mode switching and activity reduction. Such optimization would require system support and integration of KLEM into the application and OS. An extension that is already planned is to build KLEM into CogTool in order to allow automatic energy modeling using energy profiles obtained from benchmark sets defined in Section 6.1.2. Given that CogTool currently supports modeling rich interaction designs and specifying model parameters (e.g. system response time), the effort

to integrate KLEM into CogTool should be minimum – the analyst enters the type, estimated duration, and energy profiles of system activities associated with each UI event triggering operator (e.g. MOTOR and SAY) in CogTool, which can then automatically calculate the task time and energy. The most demanding work would be to identify system activities and to obtain energy profiles.

Finally, as discussed in Section 8.1, the mobile web opens great opportunities to facilitate user access to information anyplace, anytime, and could rapidly become the dominant mobile application. KLEM should thus be extended to not only take into account the interaction and computation on the device side, but also the response delays and energy expenses spent on the communication. It is very important for system designers to understand the end-to-end user performance and energy consumption.

Bibliography

[1] ACPI, Advanced Configuration & Power Interface. http://www.acpi.info.

[2] The ACT-R research group. http://act-r.psy.cmu.edu/.

[3] The CogTool project. http://www.cogtool.org.

[4] Remote Desktop Protocol (RDP) features and performance. http://www.microsoft.com/technet/prodtechnol/win2kts/evaluate/featfunc/rdpfperf.mspx.

[5] Intel corporation and Microsoft corporation. Advanced Power Management (APM) BIOS interface specification, 1996.

[6] User interface design update. http://www.keller.com/articles/readingspeed.html, 2000.

[7] Devices using OLED displays. http://www.oled-info.com/devices.html, 2007.

[8] Quarterly OLED shipment and forecast report. http://www.displaysearch.com, 2007.

[9] G. D. Abowd, C. G. Atkeson, J. Brotherton, T. Enqvist, P. Gulley, and J. LeMon. Investigating the capture, integration and access problem of ubiquitous computing in an educational setting. In *SIGCHI Conference on Human Factors in Computing Systems*, 1998.

[10] A. Acquaviva, L. Benini, and B. Ricco. Energy characterization of embedded real-time operating systems. In *Compilers and operating systems for low power*, pages 53–73. Kluwer Academic Publishers, Norwell, MA, USA, 2003.

[11] M. Anand, E. B. Nightingale, and J. Flinn. Ghosts in the machine: Interfaces for better power management. In *Proceedings of the USENIX/ACM International Conference on Mobile Systems, Applications, & Services (MobiSys'04)*, pages 23–35, 2004.

[12] J. R. Anderson and C. Lebiere. *The Atomic Component of Thought*. Lawrence Erlbaum, 1998.

[13] O. Bälter. Keystroke level analysis of email message organization. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI'00)*, pages 105–112, 2000.

[14] N. Banerjee, A. Rahmati, M. Corner, S. Rollins, and L. Zhong. Users and batteries: interactions and adaptive energy management in mobile systems. In *Proceedings of the International Conference o Ubiquitous Computing (UbiComp)*, September 2007.

[15] R. A. Baratto, J. Nieh, and L. Kim. THINC: A remote display architecture for thin-client computing. Technical report, Department of Computer Science, Columbia University, 2004.

[16] K. Barr and K. Asanovic. Energy aware lossless data compression. In *Proceedings of the USENIX/ACM International Conference on Mobile Systems, Applications, & Services (MobiSys'03)*, pages 231–244, 2003.

[17] K. Baynes, C. Collins, E. Fiterman, B. Ganesh, P. Kohout, C. Smit, T. Zhang, and B. Jacob. The performance and energy consumption of three embedded real-time operating systems. In *Proceedings of the ACM International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 203–210, 2003.

[18] G. Blasko, S. Feiner, and F. Coriand. Exploring interaction with a simulated wrist-worn projection display. In *The Ninth IEEE International Symposium on Wearable Computers (ISWC '05)*, 2005.

[19] M. Blattner, D. Sumikawa, and R. Greenberg. Earcons and icons: their structure and common design principles. *Human-Computer Interaction*, 4(1):11–44, 1989.

[20] L. Bloom, R. Harter, E. Geelhoed, M. Manahan, and P. Ranganathan. Investigating the relationship between battery life and user acceptance of dynamic, energy-aware interfaces on handhelds. In *The 6th International Symposium of Mobile Human-Computer Interaction (MobileHCI)*, 2004.

[21] S. Brewster. Overcoming the lack of screen space on mobile computers. *Personal and Ubiquitous Computing*, 6(3):188–205, 2002.

[22] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *27th Int. Symp. computer Architecture(ISCA)*, pages 83–94, 2000.

[23] T. D. Burd and R. W. Brodersen. Design issues for dynamic voltage scaling. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED '00)*, 2000.

[24] X. Cao and S. Zhai. Modeling human performance of pen stroke gestures. In *Proceedings of the SIGCHI conference on human factors in computer systems*, 2007.

[25] S. K. Card, T. P. Moran, and A. Newell. The keystroke-level model for user performance time with interactive systems. *Communications of the ACM archive*, 23(7):396–410, 1980.

[26] S. K. Card, T. P. Moran, and A. Newell. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, 1983.

[27] A. Chandrakasan, W. J. Bowhill, and F. Fox, editors. *Design of High-Performance Microprocessor Circuits*. Wiley-IEEE Press, 2001.

[28] A. P. Chandrakasan and R. W. Brodersen, editors. *Low-Power CMOS Design*. Wiley-IEEE Press, 1998.

[29] W.-C. Cheng and C.-F. Chao. Perception-guided power minimization for color sequential

displays. In *Proceedings of Great Lake Symposium on VLSI*, 2006.

[30] W.-C. Cheng, Y. Hou, and M. Pedram. Power minimization in a backlit TFT-LCD display by concurrent brightness and contrast scaling. In *Proceedings of the Conference on Design, Automation and Test in Europe*, 2004.

[31] I. Choi, H. Shim, and N. Chang. Low-power color TFT LCD display for handheld embedded systems. In *International Symposium on Low Power Electronics and Design (ISLPED)*, 2002.

[32] T. L. Cignetti, K. Komarov, and C. S.Ellis. Energy estimation tools for the Palm. In *ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, pages 96–103, August 2000.

[33] E. Clarkson, J. Clawson, K. Lyons, and T. Starner. An empirical study of typing rates on mini-QWERTY keyboards. In *Preceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '05), Extended Abstracts*, pages 1288–1291, 2005.

[34] A. Cockburn and A. Siresena. Evaluating mobile text entry with the Fastap keypad. In *People and Computers XVII (Volume 2): Proceedings of the 17th Annual British Computer Society Conference on Human-Computer Interaction (BCS HCI 2003)*, pages 77–80, 2003.

[35] S. Consolvo and M. Walker. Using the experience sampling method to evaluate ubicomp applications. *IEEE Pervasive Computing*, 2(2):24–31, 2003.

[36] L. L. Constantine and L. A. D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design (ACM Press)*. Addison-Wesley Professional, 1999.

[37] A. L. Cox and A. Walton. Evaluating the viability of speech recognition for mobile text entry. In *HCI 2004: Design For Life*, 2004.

[38] A. Dalton and C. Ellis. Sensing user intention and context for energy management. In *Workshop on Hot Topics in Operating Systems (HOTOS)*, 2003.

[39] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. J. Irwin. DRAM energy management using software and hardware directed power mode control. In *Proceedings of the 7th International Conference on High Performance Computer Architecture*, 2001.

[40] R. P. Dick, G. Lakshminarayana, A. Raghunathan, and N. K. Jha. Power analysis of embedded operating systems. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 312–315, 2000.

[41] A. Dillon, C. McKnight, and J. Richardson. Reading from paper versus reading from screens. *The Computer Journal*, 31(5):457–464, 1988.

[42] F. Douglis, R. Cáceres, F. Kaashoek, K. Li, B. Marsh, and J. Tauber. Storage alternatives for mobile computers. In *Proceedings of the 1st USENIX Symposium on Operating System Design and Implementation (OSDI)*, 1994.

[43] M. D. Dunlop and A. Crossan. Predictive text entry methods for mobile phones. *Personal and Ubiquitous Computing*, 4(2-3):134–143, 2000.

[44] C. S. Ellis. The case for higher-level power management. In *Proceedings of the 7th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VII)*, pages 162–167, 1999.

[45] K. I. Farkas, J. Flinn, G. Back, D. Grunwald, and J.-A. M. Anderson. Quantifying the energy consumption of a pocket computer and a java virtual machine. In *Measurement and Modeling of Computer Systems*, pages 252–263, 2000.

[46] Y. Fei, L. Zhong, and N. K. Jha. An energy-aware framework for coordinated dynamic software management in mobile computers. In *Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, pages 306–317, 2004.

[47] K. Flautner, S. Reinhardt, and T. Mudge. Automatic performance setting for dynamic voltage scaling. In *Proceedings of the ACM Annual International Conference on Mobile Computing & Networking*, pages 260–271, 2001.

[48] M. D. Fleetwood, M. D. Byrne, P. Centgraf, K. Q. Dudziak, B. Lin, and D. Mogilev. An evaluation of text entry in Palm OS-Graffiti and the virtual keyboard. In *Proceedings of the Human Factors and Ergonomics Society 46th Annual Meeting*, 2002.

[49] J. Flinn, K. I. Farkas, and J. Anderson. Power and energy characterization of the Itsy pocket computer (Version 1.5). Technical Report TN-56, Western Research Lab, Compaq Computer Corporation, February 2000.

[50] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *Symposium on Operating Systems Principles (SOSP)*, 1999.

[51] J. Flinn and M. Satyanarayanan. PowerScope: a tool for profiling the energy usage of mobile applications. In *Second IEEE Workshop on Mobile Computing Systems and Applications*, pages 2–10, 1999.

[52] J. Flinn and M. Satyanarayanan. Managing battery lifetime with energy-aware adaptation. *ACM Transactions on Computer Systems (TOCS)*, 22(2):137–179, 2004.

[53] S. R. Forrest. The road to high efficiency organic light emitting devices. *Organic Electronics*, 4(2-3):45–48, 2003.

[54] H. Franco, J. Zheng, J. Butzberger, F. Cesari, M. Frandsen, J. Arnold, V. R. R. Gadde, A. Stolcke, and V. Abrash. Dynaspeak: SRI's scalable speech recognizer for embedded and mobile systems. In *Proceedings of the second international conference on Human Language Technology Research*, pages 25–30, 2002.

[55] J. Froehlich, M. Y. Chen, S. Consolvo, B. Harrison, and J. A. Landay. MyExperience: a system for in situ tracing and capturing of user feedback on mobile phones. In *Proceedings of the 5th International Conference on Mobile Systems, Applications and Services (MobiSys '07)*, pages 57–70, 2007.

[56] F. Gatti, A. Acquaviva, L. Benini, and B. Ricco. Low power control techniques for TFT

LCD displays. In *International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, 2002.

[57] W. Gaver. The SonicFinder: an interface that uses auditory icons. *Human-Computer Interaction*, 4(1):67–94, 1989.

[58] D. Goldberg and A. Goodisman. Stylus user interfaces for manipulating text. In *Proceedings of the 4th Annual ACM Symposium on User Interface Software and Technology (UIST 1991)*, 1991.

[59] D. Goldberg and C. Richardson. Touch-typing with a stylus. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '93)*, 1993.

[60] J. D. Gould, L. Alfaro, R. Finn, B. Haupt, and A. Minuto. Reading from CRT displays can be as fast as reading from paper. Technical Report RC 11709 (52588), IBM Research Report, 1986.

[61] W. D. Gray, B. E. John, and M. E. Atwood. Project Ernestine: A validation of GOMS for prediction and explanation of real-world task performance. *Human-Computer Interaction*, 8(3):209–237, 1993.

[62] D. Grunwald, P. Levis, K. I. Farkas, C. B. M. III, and M. Neufeld. Policies for dynamic clock scheduling. In *Proceedings of the Symposium on Operating Systems Design & Implementation*, pages 73–86, 2000.

[63] S. Gurumurthi, A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, M. T. Kandemir, T. Li, and L. K. John. Using complete machine simulation for software power estimation: The SoftWatt approach. In *Int. Symp. High Performance Computer Architecture(HPCA)*, pages 141–150, 2002.

[64] W. J. Hansen and C. Haas. Reading and writing with computers: a framework for explaining differences in performance. *Communications of the ACM*, 31(9):1080–1089, 1988.

[65] B. L. Harrison, K. P. Fishkin, G. Anuj, C. Mochon, and R. Want. Squeeze me, hold me, tilt me! an exploration of manipulative user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computer Systems (CHI '98)*, pages 17–24, 1998.

[66] T. Harter, S. Vroegindeweij, E. Geelhoed, M. Manahan, and P. Ranganathan. Energy-aware user interfaces: An evaluation of user acceptance. In *SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 2004.

[67] P. Haunold and W. Kuhn. A keystroke level analysis of a graphics application: manual map digitizing. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI'94)*, pages 337–343, 1994.

[68] K. Hinckley, J. Pierce, M. Sinclair, and E. Horvitz. Sensing techniques for mobile interaction. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology (UIST 2000)*, pages 91–100, 2000.

[69] P. Holleis, D. Kern, and A. Schmidt. Integrating user performance time models in the design of tangible UIs. In *CHI '07 Extended Abstracts on Human Factors in Computing*

Systems, pages 2423–2428, 2007.

[70] P. Holleis, F. Otto, H. Hussmann, and A. Schmidt. Keystroke-level model for advanced mobile phone interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '07)*, pages 1505–1514, 2007.

[71] A. Holzinger. *Lecture Notes in Computer Science*, chapter Finger Instead of Mouse: Touch Screens as a Means of Enhancing Universal Access, pages 387–397. Springer Berlin / Heidelberg, 2003.

[72] Y. How and M.-Y. Kan. Optimizing predictive text entry for short message service on mobile phones. In *Proceedings of Human Computer Interfaces International (HCII '05)*, 2005.

[73] D. Huggins-Daines, M. Kumar, A. Chan, A. W. Black, M. Ravishankar, and A. I. Rudnicky. Pocketsphinx: A free, real-time continuous speech recognition system for hand-held devices. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2006)*, 2006.

[74] S. S. Intille, E. M. Tapia, J. Rondoni, J. Beaudin, C. Kukla, S. Agarwal, L. Bao, and K. Larson. Tools for studying behavior and technology in natural settings. In *Proceedings of Ubiquitous Computing (UbiComp), volume LNCS 2864*, 2003.

[75] M. Irwin, M. Kandemir, N. Vijaykrishnan, and A. Sivasubramaniam. A holistic approach to system level energy optimization. In *Proceedings of the International Workshop on Power and Timing Modeling, Optimization, and Simulation*, 2000.

[76] S. Iyer, L. Luo, R. Mayo, and P. Ranganathan. Energy-adaptive display system designs for future mobile environments. In *Proceedings of the First International Conference on Mobile Systems, Applications, and Services (MobiSys '03)*, 2003.

[77] C. L. James and K. M. Reischel. Text input for mobile devices: comparing model prediction to actual performance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 2001.

[78] B. E. John. Extensions of GOMS analyses to expert performance requiring perception of dynamic visual and auditory information. In *Preceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '90)*, 1990.

[79] B. E. John, L. J. Bass, M.-I. Sanchez-Segura, and R. J. Adams. Bringing usability concerns to the design of software architecture. In *Proceedings of The 9th IFIP Working Conference on Engineering for Human-Computer Interaction and the 11th International Workshop on Design, Specification and Verification of Interactive Systems*, 2004.

[80] B. E. John and D. E. Kieras. The GOMS family of user interface analysis techniques: Comparison and contrast. *ACM Transactions on Computer-Human Interaction*, 3(4):320–351, 1996.

[81] B. E. John and D. E. Kieras. Using GOMS for user interface design and evaluation: which technique? *ACM Transactions on Computer-Human Interaction (TOCHI)*, 3(4):287–319,

1996.

[82] B. E. John, K. Prevas, D. D. Salvucci, and K. Koedinger. Predictive human performance modeling made easy. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI'04)*, pages 455–462, 2004.

[83] B. E. John, A. Vera, M. Matessa, M. Freed, and R. Remington. Automating CPM-GOMS. In *Preceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '02)*, 2002.

[84] N. Kamijoh, T. Inoue, C. M. Olsen, M. T. Raghunath, and C. Narayanaswami. Energy trade-offs in the IBM Wristwatch computers. In *Proceedings of the Fifth International Symposium on Wearable Computers (ISWC)*, 2001.

[85] H. Kawamoto. The history of liquid-crystal displays. *Proceedings of the IEEE*, 90(4):460–500, 2002.

[86] S. Kiaei and S. Devadas. Which has greater potential power impact: High-level design and algorithms or innovative low power technology? (panel). In *Proceedings of the 1996 International Symposium on Low Power Electronics and Design*, page 175, 1996.

[87] D. E. Kieras. Towards a practical GOMS model methodology for user interface design. In *The handbook of human-computer interaction*, pages 135–158. Amsterdam: North-Holland, 1988.

[88] D. E. Kieras. Guide to GOMS model usability evaluation using NGOMSL. In *The Handbook of Human-Computer Interaction 2nd ed.* North-Holland, Amsterdam., 1996.

[89] B. Klass, D. E. Thomas, H. Schmit, and D. F. Nagle. Modeling inter-instruction energy effects in a digital signal processor. In *Proceedings of the Power-Driven Microarchitecture Workshop*, 1998.

[90] K. R. Koedinger, V. Aleven, and N. Heffernan. Toward a rapid development environment for cognitive tutors. In *Proceedings of the 11th International Conference on Artificial Intelligence in Education (AIED '03)*, pages 455–457, 2003.

[91] H. H. Koester and S. P. Levine. Validation of a keystroke-level model for a text entry system used by people with disabilities. In *Proceedings of the first annual ACM conference on Assistive technologies (Assets'94)*, pages 115–122, 1994.

[92] T. K. Landauer. *The Trouble with Computers: Usefulness, Usability, and Productivity*. The MIT Press; New Ed edition, 1996.

[93] A. R. Lebeck, X. Fan, H. Zeng, and C. S. Ellis. Power aware page allocation. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IX)*, 2000.

[94] D. Levy. The Fastap keypad and pervasive computing. In *Proceedings of the First International Conference on Pervasive Computing*, pages 58–68, 2002.

[95] K. Li, R. Kumpf, P. Horton, and T. E. Anderson. Quantitative analysis of disk drive power management in portable computers. Technical Report CSD-93-779, University of

California, Berkeley, 1994.

[96] Q. Li and F. Li. FCE: A fast content expression for server-based computing. In *IEEE International Conference on Communications*, 2004.

[97] Z. Li, C. Wang, and R. Xu. Computation offloading to save energy on handheld devices: a partition scheme. In *Proceedings of the ACM International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, 238-246, 2001.

[98] J. Lorch. A complete picture of the energy consumption of a portable computer. Master's thesis, Computer Science Department, University of California at Berkeley, 1995.

[99] J. R. Lorch and A. J. Smith. Scheduling techniques for reducing processor energy use in MacOS. *Wireless Networks*, 3(5):311–324, 1997.

[100] J. R. Lorch and A. J. Smith. Apple Macintosh's energy consumption. *IEEE Micro*, 18(6):54–63, 1998.

[101] J. R. Lorch and A. J. Smith. Improving dynamic voltage scaling algorithms with PACE. In *Proceedings of the ACM International Conference on Measurement & Modeling of Computer Systems*, pages 50–61, 2001.

[102] J. R. Lorch and A. J. Smith. Using user interface event information in dynamic voltage scaling algorithms. In *Proceedings of the IEEE/ACM International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunications Systems*, pages 46–55, 2003.

[103] Y.-H. Lu, T. Simunic, and G. D. Micheli. Software controlled power management. In *Proceedings of the 7th International Workshop on Hardware/Software Codesign*, pages 157–161, 1999.

[104] K. Lyons, D. Plaisted, and T. Starner. Expert chording text entry on the Twiddler one-handed keyboard. In *Proceedings of the Eighth International Symposium on Wearable Computers (ISWC '04)*, pages 94–101, 2004.

[105] D. Manes, P. Green, and D. Hunter. Prediction of destination entry and retrieval times using keystroke-level models. Technical Report UMTRI-96-37, University of Michigan, 1996.

[106] B. Marsh, F. Douglis, and P. Krishnan. Flash memory file caching for mobile computers. In *Proceedings of the 27th Hawaii Conference on Systems Sciences*, pages 451–460, 1994.

[107] S. Mohapatra, R. Cornea, N. Dutt, A. Nicolau, and N. Venkatasubramanian. Integrated power management for video streaming to mobile handheld devices. In *Proceedings of the Eleventh ACM International Conference on Multimedia*, pages 582–591, 2003.

[108] R. Mori, T. Matsunobe, and T. Yamaoka. A task operation prediction time computation based on GOMS-KLM improved for the cellular phone and the verification of that validity. In *6th Asian Design International Conference (ADC '03)*, 2003.

[109] A. Muttreja, S. Ravi, A. Raghunathan, and N. K. Jha. Automated performance/energy macromodeling of embedded software. In *Proceedings of the ACM/IEEE Design Au-*

tomation Conference, pages 99–102, 2004.

[110] B. A. Myers. Using hand-held devices and PCs together. *Communications of the ACM*, 44(11):34–41, 2001.

[111] R. Myung. Keystroke-level analysis of Korean text entry methods on mobile phones. *International Journal of Human-Computer Studies*, 60(5-6):545–563, 2004.

[112] J. Nielsen and R. L. Mack, editors. *Usability Inspection Methods*. John Wiley and Sons, 1994.

[113] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker. Agile application-aware adaptations for mobility. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 276–287, 1997.

[114] J. R. Olsen and G. M. Olson. The growth of cognitive modeling in human-computer interaction since GOMS. *Human-Computer Interaction*, 5(2&3):221–265, 1990.

[115] J. S. Olson and T. P. Moran. Mapping the method muddle: guidance in using methods for user interface design. In *Proceedings of a Workshop on Human-Computer Interface Design: Success Stories, Emerging Methods, and Real-World Context*, pages 269–300, 1996.

[116] M. Othman and S. Hailes. Power conservation strategy for mobile computers using load sharing. *Mobile Computing and Communications Review*, 2(1):19–26, 1998.

[117] S. Oviatt. Mutual disambiguation of recognition errors in a multimodal architecture. In *Proceedings of the SIGCHI Conference on Human Factors in Computer Systems (CHI '99)*, pages 576–583, 1999.

[118] G. Paleologo, L. Benini, A. Bogliolo, and G. D. Micheli. Policy optimization for dynamic power management. In *Proceedings of the 35th Design Automation Conference*, pages 182–187, 1998.

[119] A. Pavlovych and W. Stuerzlinger. Model for non-expert text entry speed on 12-button phone keypads. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'04)*, pages 351–358, 2004.

[120] M. Pettitt, G. Burnett, and A. Stevens. An extended keystroke level model (KLM) for predicting the visual demand of in-vehicle information systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'07)*, pages 1515–1524, 2007.

[121] A. Pirhonen, S. Brewster, and C. Holguin. Gestural and audio metaphors as a means of control for mobile devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computer Systems (CHI '02)*, pages 291–298, 2002.

[122] I. Poupyrev, S. Maruyyama, and J. Rekimoto. Ambient touch: designing tactile interfaces for handheld devices. In *Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology (UIST 2002)*, pages 51–60, 2002.

[123] V. Raghunathan, T. Pering, R. Want, A. Nguyen, and P. Jensen. Experience with a low

power wireless mobile computing platform. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED'04)*, pages 363–368, 2004.

[124] A. Rahmati, A. Qian, and L. Zhong. Understanding human-battery interaction on mobile phones. In *Proceedigns of International Conference on Human Computer Interaction with Mobile Devices & Services (MobileHCI)*, 2007.

[125] P. Ranganathan, E. Geelhoed, M. Manahan, and K. Nicholas. Energy-aware user interfaces and energy-adaptive displays. *Computer*, 39(3):31–38, 2006.

[126] J. Rekimoto, O. Haruo, and T. Ishizawa. SmartPad: a finger-sensing keypad for mobile interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computer Systems (CHI '03), Extended Abstracts*, pages 850–851, 2003.

[127] M. Rettig. Prototyping for tiny fingers. *Communications of the ACM*, 37(4):21–27, 1994.

[128] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, 1998.

[129] A. Rudenko, P. Reiher, G. J. Popek, and G. H. Kuenning. Saving portable computer battery power through remote process execution. *SIGMOBILE Mobile Computing and Communication Review*, 2(1):19–26, 1998.

[130] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Transactions on Coputer Systems*, 2(4):277–288, 1984.

[131] D. D. Salvucci and F. J. Lee. Simple cognitive modeling in a complex cognitive architecture. In *Proceedings of the SIGCHI Conference on Human Factors in Computer Systems (CHI '03)*, pages 265–272, 2003.

[132] K. Sandler. How many mobile operating systems do we need? http://www.technewsworld.com/story/how-many-mobile-operating-systems-do-we-need-61659.html?welcome=1203499281.

[133] N. Sawhney and C. Schmandt. Nomadic radio: speech and audio interaction for contextual messaging in nomadic environments. *ACM Transactions on Computer-Human Interaction*, 7(3):353–383, 2000.

[134] S. Schlosser, J. Griffin, D. Nagle, and G. Ganger. Designing computer systems with MEMS-based storage. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX)*, pages 1–12, 2000.

[135] C. Schwesig, I. Poupyrev, and E. Mori. Gummi: user interface for deformable computers. In *Proceedings of the SIGCHI Conference on Human Factors in Computer Systems (CHI '03), Extended Abstracts*, pages 954–955, 2003.

[136] C. Seo, S. Malek, and N. Medvidovic. An energy consumption framework for distributed java-based systems. In *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE '07)*, 2007.

[137] P. Shenoy and P. Radkov. Proxy-assisted power-friendly streaming to mobile devices. In

Proceedings of the SPIE/ACM Multimedia Computing and Networking Conference, 2003.

[138] E. Shih, P. Bahl, and M. J. Sinclair. Wake on wireless: an event driven energy saving strategy for battery operated devices. In *Proceedings of the 8th Annual International Conferences on Mobile Computing and Networking (MobiCom'02)*, pages 160–171, 2002.

[139] H. Shim, N. Chang, and M. Pedram. A backlight power management framework for battery-operated multimedia systems. *IEEE Design and Test of Computers*, 21(5):388–396, 2004.

[140] B. Shneiderman. Touch screens now offer compelling uses. *IEEE Software*, 8(2):93–94, 107, 1991.

[141] D. P. Siewiorek. New frontiers of application design. *Communications of the ACM*, 45(12):79–82, 2002.

[142] M. Silfverberg, I. S. MacKenzie, and P. Korhonen. Predicting text entry speed on mobile phones. In *Preceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '00)*, pages 9–16, 2000.

[143] T. Simunic, L. Benini, P. Glynn, and G. D. Micheli. Dynamic power management for portable systems. In *Proceedings of Mobile Computing and Networking (MobiCom '00)*, 2000.

[144] T. Simunic, L. Benini, P. Glynn, and G. D. Micheli. Event-driven power management. *IEEE Transaction on Computer-Aided Design of IC & Systems*, 20(7):840–857, 2001.

[145] T. Simunic, L. Benini, and G. D. Micheli. Energy-efficient design of battery powered embedded systems. In *Proceedings of the 1999 International Symposium on Low Power Electronics and Design*, pages 212–217, 1999.

[146] A. Sinha and A. Chandrakasan. JouleTrack - a web based tool for software energy profiling. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 220–225, 2001.

[147] A. Sinha, A. Wang, and A. Chandrakasan. Algorithmic transforms for efficient energy scalable computation. In *Proceedings of the IEEE International Symposium on Low-Power Electronic Design (ISLPED'00)*, 2000.

[148] C. U. Smith. *Performance Engineering of Software Systems*. Addison-Wesley, 1990.

[149] C. U. Smith and L. G. Willams. *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison-Wesley Professional, 2002.

[150] C. U. Smith and L. G. Williams. Best practices for software performance engineering. In *Proceedings of CMG*, 2003.

[151] M. Stemm and R. H. Katz. Measuring and reducing energy consumption of network interfaces in hand-held devices. *IEICE Transactions on Communications*, E80-B(8):1125–1131, 1997.

[152] T. K. Tan, A. Raghunathan, and N. K. Jha. EMSIM: an energy simulation framework

for an embedded operating system. In *Proceedings of the International Symposium on Circuits & Systems*, pages 464–467, 2002.

[153] T. K. Tan, A. Raghunathan, and N. K. Jha. Energy macromodeling of embedded operating systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 4(1):231–254, 2005.

[154] T. K. Tan, A. Raghunathan, G. Lakshminarayana, and N. K. Jha. High-level software energy macro-modeling. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 605–610, 2001.

[155] V. Tiwari, S. Malik, and A. Wolfe. Compilation techniques for low energy: an overview. In *Proceedings of the 1994 Symposium on Low Power Electronics*, 1994.

[156] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: a first step towards software power minimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2(4):437–445, December 1994.

[157] S. Udani and J. Smith. The Power Broker: intelligent power management for mobile computers. Technical Report MS-CIS-96-12, Department of Computer Science, University of Pennsylvania, 1996.

[158] A. Vahdat, A. R. Lebeck, and C. S. Ellis. Every joule is precious: a case for revisiting operating system design for energy efficiency. In *Proceedings of the 9th ACM SIGOPS European Workshop*, 2000.

[159] K. S. Vallerio, L. Zhong, and N. K. Jha. Energy-efficient graphical user interface design. *IEEE Transactions on Mobile Computing*, 7(5):846–859, 2006.

[160] N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. S. Kim, and W. Ye. Energy-driven integrated hardware-software optimizations using SimplePower. In *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA)*, pages 95–106, 2000.

[161] H. Vrsalovic, M. Hornyak, L. Luo, D. P. Siewiorek, and A. Smailagic. A computer system for accessing ambient display and computing resources in wearable environments. In *Proceedings of the 10th IEEE International Symposium on Wearable Computers*, 2006.

[162] A. Waibel, A. Badran, A. W. Black, R. Frederking, D. Gates, A. Lavie, L. Levin, K. Lenzo, L. M. Tomokiyo, J. Reichert, T. Schultz, D. Wallace, M. Woszczyna, and J. Zhang. Speechalator: two-way speech-to-speech translation in your hand. In *Proceedings of the European Conference on Speech Communication and Technology*, 2003.

[163] L. Weberg, T. Brange, and A. W. Hasson. A piece of butter on the PDA display. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '01), Extended Abstract*, 2001.

[164] D. Wigdor and R. Balakrishnan. TiltText: using tilt for text input to mobile phones. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology (UIST 2003)*, pages 81–90, 2003.

[165] C. Xian and Y.-H. Lu. Energy reduction by workload adaptation in a multi-process envi-

ronment. In *Design Automation and Test in Europe*, 2006.

[166] C. Xian, Y.-H. Lu, and Z. Li. A programming environment with runtime energy characterization for energy-aware applications. In *Proceedings of the 2007 International Symposium on Low Power Electronics and Design*, pages 141–146, August 2007.

[167] S. Xiong, W. Xie, Y. Zhao, J. Wang, E. Liu, and C. Wu. A simple and flexible driver for OLED. In *Proceedings of the Asian Symposium on Information Display (ASID '99)*, pages 147–150, 1999.

[168] Y. Xiong, X. Zhou, X. Li, and Y. Gong. OOEM: object-oriented energy model for embedded software reuse. In *IEEE International Conference on Information Reuse and Integration (IRI '03)*, pages 551–558, 2003.

[169] W. Ye, N. Vijaykrishna, M. Kandemir, and M. Irwin. The design and use of SimplePower: a cycle-accurate energy estimation tool. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 340–345, 2000.

[170] G. K. Yeap. *Practical Low Power Digital VLSI Design*. Springer, 1997.

[171] J. Zedlewski, S. Sobti, N. Garg, F. Zheng, A. Krishnamurthy, and R. Wang. Modeling hard-disk power consumption. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST '03)*, pages 217–230, 2003.

[172] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. ECOSystem: managing energy as a first class operating system resource. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, 2002.

[173] L. Zhong and N. K. Jha. Graphical user interface energy characterization for handheld computers. In *Proceedings of International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, 2003.

[174] L. Zhong and N. K. Jha. Dynamic power optimization of interactive systems. In *Proceedings of the International Conference on VLSI Design*, 2004.

[175] L. Zhong and N. K. Jha. Energy efficiency of handheld computer interfaces: limits, characterization, and practice. In *Proc. USENIX/ACM Int. Conf. Mobile Systems, Applications, and Services (MobiSys)*, 2005.