BOOTSTRAPPING LOCATION-AWARE PERSONAL COMPUTING

BY NISHKAM RAVI

A dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Computer Science

Written under the direction of

Liviu Iftode

and approved by

_____

_____

_____

_____

New Brunswick, New Jersey

January, 2008

ABSTRACT OF THE DISSERTATION

Bootstrapping Location-aware Personal Computing

by Nishkam Ravi

Dissertation Director: Liviu Iftode

Pervasive computing is centered around the idea of provisioning computing services to the user anywhere anytime. If realized, pervasive computing can have a significant impact on our daily lives, ranging from the the way we dress to the way we work and travel. Two key hurdles prohibit the widescale adoption of pervasive computing. First, human cognitive bandwidth is a limited resource; the burden of interacting with pervasive computing applications may outweigh the functionality obtained. Second, pervasive computing applications often assume a smart environment which does not exist today; dependency on a ubiquitous computing infrastructure hampers deployment.

In this dissertation, we propose *location-aware personal computing* as a way of getting close to the pervasive computing vision with minimal overhead. Central to location-aware personal computing is the use of smart phones and location information. Smart phones personify a ubiquitous personal device that can execute client frontends, and connect wirelessly to backend services. Location information serves as a proxy for the user. Smart phones and location information can minimize both user involvement and dependency on a ubiquitous computing infrastructure.

We investigate the challenges in bootstrapping location-aware personal computing, namely ad-hoc service provisioning, infrastructureless location determination, power management and location privacy. We present solutions for each and comment on how

they can bootstrap location-aware personal computing.

Provisioning locally embedded services to the user without prior knowledge of the environment is an important aspect of location-aware personal computing. This dissertation proposes the use of dual connectivity (i.e Bluetooth and 3G) on smart phones for discovering and provisioning local services to the user without any pre-configuration. A novel service provisioning protocol, called SDIPP (Service Discovery, Interaction and Payment Protocol), is presented and evaluated. SDIPP is implemented on top of a middleware called Portable Smart Messages. The design of Portable Smart Messages is presented. Experimental results show that SDIPP can discover and provision locally embedded services to the user within acceptable time limits.

In order to enable location-aware personal computing, continuous location updates are needed both indoors and outdoors. Determining user location without installing extra infrastructure is a hard problem. This dissertation proposes use of light sensors and cameras on phones for sensing user location in indoor environments without requiring any infrastructure support. Low-level sensory input is converted into location information using simple vision and classification algorithms. Experiments show that location can be determined with fairly high room-level accuracy.

The use of resource-constrained personal devices, such as smart phone, is central to location-aware personal computing. It is important to manage the limited resources on these devices. The most crucial resource is battery lifetime. This dissertation shows how location information can aid in battery management with smart phones as the case study. By storing past location traces of the user on the smart phone, future whereabouts of the user can be predicted. This information can be useful in estimating when the next opportunity for charging the phone will be encountered. The knowledge of when the user will charge the phone next can be instrumental in managing the limited battery lifetime on smart phones across multiple applications.

Location-aware personal computing requires user location to be shared with untrusted third parties, such as web services. Releasing location information without breaching user privacy is a hard problem. This dissertation proposes a new information-flow control model, called Non-inference, for protecting location privacy of the user

against untrusted services without compromising the quality of service. It is shown that Non-inference is undecidable in general but decidable for programs that satisfy uni-directional information flow. Non-inference is decided using static program analysis techniques.

The main conclusion of this research is that the cooperative use of smart phones and user location can be instrumental in devising low-cost, easy-to-use and non-intrusive solutions for pervasive computing without requiring significant infrastructure support.

Acknowledgements

I would like to thank Liviu Iftode, my advisor, for his guidance, friendship, and unflinching support. I am grateful to him for giving me the freedom to work on my own ideas and for bearing with me for my unruly work schedule. I thank the members of my dissertation committee: Prof. Badri Nath, Prof. Ahmed Elgammal and Prof. Tatsuo Nakajima for their helpful comments and feedback on my thesis. I thank Profs. Michael Littman, D.M. Dhamdhere and several other researchers that I truly respect, for passing on to me their research values, which will stay with me throughout my research career.

I would like to thank all my colleagues and friends for keeping the journey interesting. The list is endless. I am grateful to my family for believing in me, and for their love, support and patience throughout these years. Finally, I would like to thank Rutgers for being a great home.

Table of Contents

Abstract ii

Acknowledgements v

List of Tables ix

List of Figures x

1. Introduction 1

1.1. Thesis 1

1.2. Location-aware Personal Computing 1

1.3. Seamless Service Discovery and Provisioning 4

1.4. Determining User Location 7

1.5. Location-aware Battery Management 10

1.6. Location Privacy 12

1.7. Summary of Dissertation Contributions 14

1.8. Contributions to the Dissertation 15

1.9. Organization of the Dissertation 15

2. Location-aware Service Discovery and Provisioning 16

2.1. Protocols 17

2.1.1. Discovery Protocol 17

2.1.2. SDIPP Interaction Protocol 19

2.1.3. SDIPP Payment Protocol 20

2.1.4. Bootstrapping 23

2.1.5. SDIPP Architecture 23

2.2. Portable Smart Messages 23

2.2.1. Portable SM Architecture . . . 24
2.2.2. Portability . . . 26
2.3. Implementation and Evaluation . . . 33
2.3.1. Case Study . . . 33
2.3.2. Bluetooth Inquiry Clash . . . 35
2.4. Discussion . . . 38
2.5. Summary . . . 38

3. Determining User Location Indoors . . . 40
3.1. Localization Approach . . . 40
3.1.1. Location Determination with Camera Phone . . . 41
3.1.2. Location Determination with Light Sensor . . . 41
3.2. Data Collection . . . 42
3.3. Localization Algorithms . . . 44
3.3.1. Location Determination with Camera-phone . . . 45
3.3.2. Location Determination with Light Sensor . . . 49
3.4. Evaluation . . . 52
3.4.1. Experimental Results for Camera-phone Based Localization . . . 52
3.4.2. Experimental Results for Light Sensor Based Localization . . . 57
3.5. Discussion . . . 62
3.6. Summary . . . 62

4. Location-aware Battery Management . . . 64
4.1. Architecture and Algorithms . . . 66
4.1.1. Charging Opportunity Predictor . . . 67
4.1.2. Call Time Predictor . . . 68
4.1.3. Battery Lifetime Predictor . . . 69
4.1.4. Viceroy and User Interface . . . 71
4.2. Experimental Results for the Battery Management Architecture . . . 72
4.3. Discussion . . . 79

4.4. Summary 79

5. Information Flow Control for Location Privacy 80

5.1. Information Flow Control 81

5.2. Non-Inference 83

5.3. Proofs for the Theoretical Properties of Non-Inference 85

5.4. Model 90

5.5. Deciding Non-Inference for Location Based Applications Using Static Analysis 91

5.5.1. Information-Flow Relations 92

5.5.2. Construction of Information-Flow Relations 93

5.5.3. Solving Information-Flow Relations 94

5.6. Implementation and Evaluation 97

5.6.1. Discussion 102

5.6.2. Limitations 104

5.7. Summary 105

6. Summary and Conclusions 106

6.1. Directions for Research 111

6.2. The Symbian and iPhone Era 111

6.3. In Retrospect 112

References 114

Vita 120

List of Tables

1.1. *Comparison of Service Discovery Protocols* 5

1.2. *Comparison of Localization Systems* 8

2.1. *Performance Evaluation of SDIPP* 32

2.2. *Effect of Data Brick Size on Single-Hop Portable SM Round-Trip Time* 34

2.3. Comparison of *Random* With Our Policy for Minimizing Bluetooth Inquiry Clash 36

3.1. *Success Probability for the Three Localization Experiments (Using Camera-phone)* 55

3.2. *Energy Consumption and Response Time for Sending an Image and Receiving Location Update* 55

4.1. *Comparing Accuracy of Our Battery Lifetime Prediction Algorithm with ACPI's* 78

List of Figures

1.1. *Pictorial Overview of Location-aware Personal Computing* 2

1.2. *Service Discovery Overview* 4

2.1. *SDIPP Discovery Model* 17

2.2. *SDIPP Interaction Protocol* 20

2.3. *SDIPP Payment Protocol* 21

2.4. *Portable Smart Message Architecture* 24

2.5. *Portable SM Pseudo-code before Instrumentation* 29

2.6. *Portable SM Pseudo-code after Instrumentation* 30

2.7. *Example of Resuming SM Execution after Migration* 31

2.8. Policy for Timeout For Minimizing Bluetooth Inquiry Clash 36

3.1. *Left: Phone as Pendant, Right: Snapshot of Client* 42

3.2. *Room Layout* 43

3.3. *Two Low-resolution Images of the Same Corner* 43

3.4. *Left: Light Sensor on a Hat. Right: Light Sensor as a Pendant* 44

3.5. *Apparatus Used for Data Collection for Light-based Localization* 44

3.6. *System Architecture for Camera-phone Based Localization* 50

3.7. *Bayesian Fingerprints (Histograms) for Four Rooms* 51

3.8. *Range-max Fingerprints for Six Rooms* 52

3.9. *Low-Resolution Pictures of a Few Rooms Taken from the Door* 53

3.10. *Low-Resolution Query Image Matches with Image 3* 53

3.11. *Low-Resolution Pictures of Different Corners* 54

3.12. *Query Image Matches with Image 4* 54

3.13. *Left: Image in the Database; Center: Image With a Person; Right: Person Wearing a Brown Jacket* 56

3.14. *Rooms Without Windows. Light Sensor Worn atop a Hat (top) and as Pendant (bottom)* 58
3.15. *Rooms Without Windows. Light Sensor Worn atop a Hat (top) and as Pendant (bottom).* 60
3.16. *Rooms With Windows. Light Sensor Worn atop a Hat* 61
4.1. *Battery Management System Architecture* 66
4.2. *Base Curve for a New HP Laptop (top) and Old Dell Laptop (bottom)* . 70
4.3. *Base Curve for an HP iPAQ* 71
4.4. *Charging Opportunity Prediction Error for Various Sample Sizes and History Sizes* 73
4.5. *Absolute Call Time Prediction Error for Weekdays (top) and Weekends (bottom)* 74
4.6. *Cumulative distribution function of the Length of Phone Calls (top) and the Number of Calls Made During Each Hour (bottom)* 75
4.7. *Base Curve Together With Discharge Curves (actual and derived) for the New HP laptop (top) and Old Dell laptop (bottom)* 77
4.8. *Base Curve Together with Discharge Curves (actual and derived) for HP iPAQ* 78
5.1. *Framework for Applying Non-Inference* 90
5.2. *Function that Calculates Distance Between Two Cars and Average Values of Coordinates* 94
5.3. *Information-flow Relations for the Distance Example* 95
5.4. *Linear Equations for the Distance Example* 96
5.5. *Running Time of the Static Analyzer* 104

Chapter 1

Introduction

1.1 Thesis

In order to design low-cost and easy-to-use pervasive computing applications, it is necessary to minimize both human involvement and dependency on a ubiquitous computing infrastructure. Location-aware personal computing is a way of achieving the pervasive computing vision in a cost-effective fashion. Central to location-aware personal computing is the use of location information as user proxy; and smart phones as intelligent personal devices that can execute client frontends and connect wirelessly to backend services. In this dissertation, we address four key challenges in bootstrapping location-aware personal computing, namely ad-hoc service provisioning, infrastructureless location determination, power management and location privacy.

1.2 Location-aware Personal Computing

Desktop computing has, for decades, been synonymous with personal computing. Over the last decade laptop computers have played a significant role in redefining personal computing. By recognizing *mobility* as an integral part of human behavior and designing for it, laptops have challenged the notion of desktop computing as personal computing and inspired *mobile computing*– the ability to compute on-the-go. However, despite the seemingly increasing use of laptops as mobile computers, a recent study shows that they are predominantly used by users at fixed places as lightweight desktops rather than as mobile computers [84]. This is primarily due to the inability of users to carry laptops in their pockets. This is fast changing. Miniaturization has permitted the capabilities of laptop computers to be incorporated in mobile phones. This has led

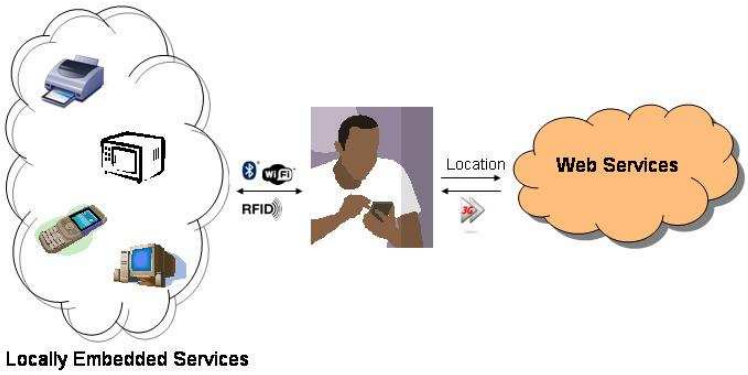

Figure 1.1: *Pictorial Overview of Location-aware Personal Computing*

to the emergence of *smart phones*, which can run applications and client frontends, communicate with other devices using WiFi/Bluetooth, connect to the internet over 3G, and sense information using camera and wireless interfaces. The ability of users to carry such powerful computing devices in their pockets has once again challenged the notion of personal computing and taken it closer to mobile computing.

The evolution of mobile devices has been paralleled by the advances in ubiquitous computing whose ultimate goal is to provide *everywhere computing*, i.e the ability to compute anywhere, anytime. Everywhere computing is characterized by users interacting with services offered by smart devices embedded everywhere in the environment. When Mark Weiser sketched out the vision of everywhere computing in 1991, in his seminal paper called "Computer for the twenty first century" [83], he overlooked two major issues: (1) human cognitive bandwidth is limited; the burden of interacting with a ubiquitous computing infrastructure may outweigh the functionality obtained, and (2) installing a ubiquitous computing infrastructure is costly and perhaps even unprofitable,

Smart phones with ample computing power, multiple wireless interfaces, always-on

internet connectivity and ability to execute applications and client frontends, have provided an alternative to everywhere computing [41]. Instead of embedding computing everywhere the environment, it can be distributed over the internet, exported as web services and accessed using smart phones. The difference between a locally embedded service and a web service is that the former is location-dependent and physically available while the later is not. If web services could be made location-aware they could customize themselves to serve as locally embedded services. This is not always possible though. For example, a local printer or display cannot be replaced by a web service. Therefore, some amount of computing (which is not replaceable by web services) would have to be embedded in the environment. In order to discover and interact with this intermittent locally embedded computing without involving the user, the smart phone carried by the user would have to be location-aware. Thus, we see location and smart phones as being crucial to this simple and viable form of computing, which we call *location-aware personal computing*. In location-aware personal computing, the ubiquitous computing infrastructure is replaced by a combination of location-aware web services (also known as *location-based services*), a limited amount of locally embedded computing, and smart phones. Location information acts as user proxy and helps in minimizing user involvement. Location-aware personal computing is a way of achieving the pervasive computing vision in a cost-effective fashion. Figure 1.1 gives a pictorial overview of location-aware personal computing.

Bootstrapping location-aware personal computing involves the following set of challenges: (1). discovering and provisioning locally embedded services to the smart phone of the user without pre-configuration, (2). determining user location without requiring extra infrastructure, (3). managing the battery lifetime of the smart phone and (4). protecting location information of the user from illegal access and misuse. In the rest of this section, we describe each of these challenges in more detail, present prior work and briefly describe my contributions.

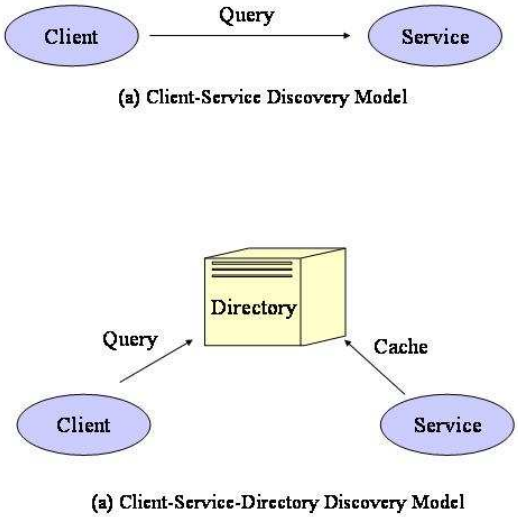

Figure 1.2: *Service Discovery Overview*

1.3 Seamless Service Discovery and Provisioning

Pervasive computing is centered around the idea of provisioning services to the user. These services could be *private services*, those established by a user or group of users for their own personal use (e.g smart home); *non-profit services*, those established by a public organization (e.g a bus information service at a bus-station); *pay-per-use services*, those established by private vendors (e.g entertainment services); and *cooperative services*, those provided by a group of people to each other (e.g vehicular networking). While many of these services can be established as web services (with proper customization often requiring location information), others have to be embedded in physical space. Protocols are needed for seamlessly discovering these locally embedded services and provisioning them to the user. These are called Service Discovery Protocols (SDP's).

Service discovery protocols typically consist of three entities: the client, the directory, and the service itself. Most SDP's follow one of the two discovery models: *client-service* model and *client-service-directory* model (as shown in Figure 1.2). In *client-service* model, services advertise themselves and clients directly query the services. In

Table 1.1: *Comparison of Service Discovery Protocols*

Protocols	Invocation	Query vs Announce	Directory based?	Service Matching	Context-aware	Scope
Bluetooth	None	Query	No	Match all	No	Vicinity
INS	None	Both	Yes	Match best	No	Admin domain
Jini	Java code	Both	Yes	Match all	No	Admin domain
Salutation	RPC	Both	Yes	Match one/all	No	Admin domain
SLP	URL	Both	Yes	Match all	No	Admin domain
UPnP	None	Both	Partially	Match all	No	Admin domain
SDIPP	Java code	Query	Yes and No	Match best	Yes	Multi-scoped

client-service-directory model, services cache their information on a centralized directory and clients query directories in order to discover services. While the *client-service* model requires less infrastructure and is more suitable for nomadic environments, it is not nearly as powerful as the *client-service-directory* model.

Several service discovery protocols have been proposed in the past. These protocols can be classified based on their properties (as shown in Table 1.1). In most of the protocols, services advertise themselves by announcing their presence using broadcast packets and by responding to query packets, with the exception of Bluetooth services, which do not announce their presence and respond only when queried. In addition, some of these protocols use directories for caching service information in order to aid in service announcement and discovery. Protocols that use service announcements and directories use more infrastructure and network bandwidth.

With the exception of Jini [81] Salutation [13] and SLP [37], none of the previously proposed protocols provide a service invocation mechanism. In Jini [81], the directory not only provides service look-up but also downloadable Java code/objects for interacting with the service using RMI. Salutation [13] on the other hand uses RPC (Remote Procedure Call) for service invocation. Service Location Protocol [37] is a lightweight protocol that targets service discovery within a site. It uses URL-based service invocation mechanism. Directories are optional.

Most of the protocols list/match all the relevant services after the discovery phase is over. Service selection, is therefore, limited or missing (with the exception of INS [18]). In addition, user's context information (such as location) is not taken into account for service selection. Finally, every protocol is limited in its scope and most work within a

pre-specified administration domain. Jini, for example, targets enterprise environments, while UPnP [15] is designed for home and office environments. UDDI [12] (not listed in the table) targets profit-based web services, and uses a web-based distributed directory for advertisement, similar to *yellow pages*.

There is a need for a hybrid approach that is wide in scope, requires minimal pre-configuration and is not limited to a pre-specified administrative domain. Also, the protocols proposed earlier overlook the payment aspect of service provisioning, which should be addressed.

Bluetooth is a wireless protocol designed for low-power consumption and short-range communication (1-100 meters). Its primary purpose is to connect devices such as mobile phones, printers, laptops and digital cameras. Bluetooth also has its own service discovery protocol. Bluetooth SDP [14] follows the client-service model and enables nearby devices to discover services on each other. Bluetooth is a low power protocol making it suitable for energy-constrained devices. Bluetooth SDP is query based, which means that clients query for available services rather than services pro-actively announcing their presence. It uses unicast and broadcast as the communication mechanism and does not provide any service invocation mechanism.

Bluetooth SDP has been incorporated in smart phones and follows the *client-service* discovery model, which requires less infrastructure and is more suitable for nomadic environments, but is not as powerful as the *client-service-directory* model. Since smart phones have always-on internet connectivity in the form of 3G/GPRS, directories can be conveniently maintained on the internet. GPRS (General Purpose Radio Service) also known as 2.5G is a mobile data service that uses wide area cellular phone networks to provide low-bandwidth internet connectivity; 3G is an enhanced and improved standard that provides higher bandwidths.

In this dissertation, we present a Service Discovery, Interaction and Payment Protocol (SDIPP) [68], which follows the *client-service-discovery* model by combining Bluetooth SDP with 3G connectivity. It provides a mechanism for service invocation without any prior knowledge of the service. This is accomplished by downloading the client-side

code for interacting with the service from a trusted web server on-the-fly. Service discovery protocols proposed earlier, overlook the payment aspect of service provisioning. SDIPP provides a safe electronic payment mechanism based on Millicent scrips [35]. In addition, SDIPP is context-aware and multi-scoped. We have implemented and tested this protocol on Sony Ericsson P900 phones. Table 1.1 compares the properties of SDIPP with the popular service discovery protocols.

1.4 Determining User Location

In order to enable location-aware personal computing, continuous location updates are needed both outdoors and indoors. Global Positioning System (GPS) is a globally accepted solution for determining user location outdoors. It uses the existing satellite infrastructure for determining user location. The user is required to carry a GPS receiver, which calculates its position by measuring and triangulating the distance between itself and three or more GPS satellites. No such low-cost and easy-to-use solution exists for indoor localization. A good indoor localization system should score well on a number of metrics including, precision, infrastructure cost, privacy and coverage. Prior work on determining user's location indoors can be broadly classified into nine categories, based on the technology used for localization: RF-based, Ultrasound-based, WiFi-based, Bluetooth-based, GSM-based, audio-based, Powerline-based, smart-floor-based and vision-based. In RF-based systems (e.g ActiveBadge [82]) an IR badge worn by the user emits a unique IR signal periodically. Sensors installed at known positions pick up the signal and update the position of the badge in a centralized database. Such systems provide room level positioning and incur significant installation and maintenance costs.

Ultrasound-based localization systems can be classified in two categories: ones that provide centimeter level positioning by requiring ultrasound receivers to be installed on ceilings and ultrasound transmitters to be carried by users (e.g Active Bat [39]), and ones that provide meter level accuracy by requiring ultrasound transmitters to be installed at known coordinates inside buildings and ultrasound receivers to be carried by users (e.g Cricket [60] and WALRUS [27]). Like RF-based systems, Ultrasound

Table 1.2: *Comparison of Localization Systems*

System	Precision	Infrastructure Cost	Privacy
ActiveBadge	Room-level	RF Sensors and Tags	Server tracks users
Active Bat	5-10 cm	Ultrasound receivers and transmitters	Server tracks users
Cricket	1 m	Ultrasound receivers and transmitters	User device computes location
Radar	5 m	WiFi beacons	User device computes location
GSM-based	5-50 m	GSM coverage	User device computes location
Place Lab	15-30 m	WiFi, GSM or Bluetooth beacons	User device computes location
Smart Floor	Sub-room level	Special floor tiles	Server tracks user
EasyLiving	Sub-room level	Cameras in rooms	Server tracks user
WALRUS	Room-level	PC per room and WiFi beacons	User device computes location
Audio Location	15-30 cm	Microphones	Server tracks users
PowerLine	Sub-room level	Signal generating modules and receivers	User device computes location
Camera-phone	Sub-room level	None	User device computes location
Light-intensity	Room-level	None	User device computes location

systems also require specialized infrastructure to be installed and thus incur significant cost of deployment.

WiFi-based systems (e.g Radar [22]) operate by recording and processing signal strength information at multiple WiFi base-stations. They use signal propagation modeling to determine user's location with up to five meters accuracy. Although no additional hardware is required, WiFi coverage is assumed. Also, WiFi-based systems cannot directly determine which room the user is in, which is important for many location-aware applications. Bluetooth-based systems [28] determine user's location upto a few meters, but require Bluetooth radios/base-stations to be installed in the environment, thus suffering from the same drawback as the other localization systems.

GSM-based localization [56], which uses wide signal-strength fingerprints, is a good solution for indoor localization because it does not require any extra infrastructure and is claimed to achieve a median accuracy of five meters. The main limitation of a GSM-based approach is that it is hard to tell which side of the wall the user is on. This is crucial for several applications. A location-aware system may connect the user's phone to the wall-display in the neighboring room, which is not acceptable. GSM-based solutions make the assumption that the user has a GSM phone, while in several countries including the US, CDMA is more popular.

Place Lab [48] is an effort at combining radio-based localization techniques. It

works by listening for the transmissions of radio sources such as WiFi access points, fixed Bluetooth devices, and GSM cell towers. A beacon database provides location information based on the IDs of beacons. PlaceLab can provide user location with up to 15 meters of accuracy. PlaceLab is a high-coverage location determination system. However, presence of beacons, corresponding receivers and beacon database is assumed.

Audio Location [75] determines user location by using microphones that listen to sounds made by the users themselves such as finger clicking. It is a low-cost system that achieves centimeter level accuracy. However, it assumes the presence of microphones in the environment and may not work properly in a noisy environment. Power-Line localization [57] achieves sub-room level localization by fingerprinting of multiple tones transmitted along power lines. Though effective, this approach requires signal-generating modules to be installed in buildings and corresponding receivers to be carried by users. Smart Floor system [55] uses special floor tiles to identify users based on their footsteps. Although the user does not have to carry any special device, the floor needs to be instrumented with these special tiles.

Microsoft's EasyLiving [47] project uses cameras installed in rooms to track humans using vision techniques. The cost of installing cameras in every room makes deployment difficult. Privacy is is a big issue as users are continuously *watched*. Work is being done in using camera phones as interaction devices by tagging physical objects with visual codes and using vision techniques to extract and interpret the information stored in these visual codes [70, 24, 69, 74, 78]. Localization could also be possibly achieved with this method. However, physical objects would have to be tagged.

With the exception of GSM-based, all other approaches require specialized devices to be installed in buildings, which increases the cost and decreases the chances of deployment. GSM-based localization has its own limitations as described earlier. Since the goal of location-aware computing is to obviate the need of a ubiquitous computing infrastructure and to minimize user involvement, the cost of the localization system and the inconvenience of using it should be negligible, otherwise the purpose is defeated.

This dissertation presents two infrastructureless solutions [67, 64] for indoor localization: camera-phone-based localization and light-intensity-based localization. These

two solutions can potentially be combined together for higher accuracies. In the camera-phone-based localization solution [67], images are periodically captured by the camera phone worn by the user as a pendant and transmitted to a web server over 3G. The web server maintains a database of images with their corresponding locations. Upon receiving an image, the web server compares it with stored images, and based on the match, estimates user's location. We accomplish this with off-the-shelf image matching algorithms, by tailoring them for our purpose. This solution does not require any infrastructure to be installed in the environment; neither custom hardware nor wireless access points are required; physical objects do not have to be "tagged" and users do not have to carry any special device. In the light-intensity-based localization solution [64], the location of the user is determined based on the intensity of light incident on a light sensor worn by the user. In the dissertation, we show that every room has a unique light-intensity fingerprint that can be used to identify it. This solution also does not require any infrastructure to be installed in the environment, and the user is only required to carry a tiny light sensor. Table 1.2 compares the popular indoor localization systems on a number of metrics.

1.5 Location-aware Battery Management

Location-aware personal computing is centered around the idea of running client frontends and applications on resource-constrained personal devices (e.g smart phone), many of which would run as background tasks including daemons for determining user's location, daemons for listening to incoming network requests, daemons for warming the phone cache etc. These processes would compete for the limited resources on these devices. The most crucial resource, which determines the availability of a personal device, is battery lifetime. We study the case of smart phone, which in this dissertation, represents the intelligent personal device.

On smart phones, the interfaces that inform the user of the battery levels (such as ACPI [1]) have not kept up with the evolution of the capabilities of these devices. A simple "battery remaining" or even "time remaining" does not enable the user to make the right decisions as to the spend of the energy budget and the request for recharging.

Prior research on dealing with the limited battery lifetime problem has focused on optimizing energy at different levels of the stack, starting with hardware all the way up to the application layer. The most commonly used energy optimization mechanism is *hibernation*, in which the CPU or other hardware units of a system are put in low-power mode when not being actively used. This is commonly done by the operating system. Dynamic voltage scheduling is another commonly used optimization technique in which a compiler-directed algorithm identifies program regions where the CPU can be slowed down with negligible performance loss [46, 40]. Two effective application-level energy optimizations have been proposed: cyberforaging [23] and application adaptation [32, 20]. In cyberforaging, computation is off-loaded from the battery powered mobile device to a wall-powered server whenever profitable. In application adaptation, the fidelity of an application is lowered whenever low battery levels are detected. While a lot of work has been done on optimizing energy, there has been limited research on managing battery lifetime across applications and treating energy as a first-class operating system resource. To the best of our knowledge, *ECOSystem* [86] is the only piece of work that takes this approach. ECOSystem is a framework for sharing battery lifetime as a resource across competing applications.

Location information has never been exploited in managing battery lifetime. In this chapter, we describe a location-aware battery management scheme [66, 63] for smart phones, which is based on two key observations. First, it is necessary to create an energy budget for different applications in order to prevent low priority applications (such as background tasks) from affecting the availability of high priority applications (such as telephony). Second, the knowledge of when the user will recharge the phone next is crucial to managing battery lifetime on phones. This is because the knowledge of when the next recharge will happen determines the total battery lifetime available to applications. User studies show that users typically charge their phones at fixed locations [25]. Hence, by predicting the whereabouts of the user, it should be possible to predict charging opportunities. We present a system architecture for logging user information (such as location, call duration, etc) on the phone and a set of algorithms for making predictions based on these logs. This allows the system to determine the

amount of battery lifetime needed for safe execution of crucial applications and to warn the user if one or more non-crucial applications need to be terminated.

1.6 Location Privacy

Extensive deployment of location-aware computing endangers user's location privacy and exhibits significant potential for abuse. The US government realized the seriousness of the this problem and released the *Location Privacy Protection Act* [8] in 2001. Unless the users are secure about their location privacy, they would be reluctant to use location-aware applications. This hurdle would have to be overcome if location-aware computing is to be globally accepted.

Releasing location information to untrusted parties in a privacy-friendly manner is a challenging task. Early work on context privacy (which includes location) focussed mostly on a policy-based approach [31, 43]. Policies could be of one of the following two kinds: those specified by the service providers and those specified by the users. The policies served as a mutual agreement on the manner in which data was to be collected, shared and used. The main problem with this method was that the enforcement of these policies was loosely defined. The users essentially had to trust the service providers for abiding by these policies.

The use of policy-based approach was followed by the application of k-anonymity [77, 36] to location information using data perturbation techniques. Location information was depersonalized and perturbed before being forwarded to the LBS. This was accomplished by reducing the spatiotemporal resolution (using a quadtree-based algorithm) until the data met the k-anonymity [77, 36] constraint. A subject is considered k-anonymous if it cannot be distinguished from at least k - 1 other subjects. For example, if the location information sent by a mobile subject is perturbed to replace the exact coordinates by a spatial interval, such that the locations of at least k - 1 other subjects belong to that interval, then the adversary cannot match the location of the subject to its identity without a certain amount of uncertainty. The uncertainty would increase with k, providing better privacy. This method is the state of the art in location

privacy. The main drawback of this method is that it may lead to inferior quality of service where accurate location information is desired. Also, it uses a global value of k, which is decided statically, where a smaller value of k could provide desired privacy levels without affecting quality of service significantly.

This dissertation proposes an information-flow control based approach for location privacy, which preserves quality of service. Information-flow control policies tend to impose restrictions on the manner in which sensitive data flows through a program/system. State of the art in information-flow control is *Non-interference* [26, 59]. Intuitively, non-interference requires that high-security information does not affect the low-security observable behavior of the system. In other words, private data does not influence public data. That is, if the value of a public variable q depends on that of a private variable p then non-interference is violated. In effect, non-interference isolates private data from public data. In doing so, it guarantees that the publicly observable behavior of a system/program that does not reveal *anything* about its private behavior. However, data isolation is an extreme measure, and in many real applications it is not possible to isolate private data from public data. As such, non-interference cannot be applied for location privacy.

This dissertation proposes a weaker model of information-flow control called *non-inference* [62]. *Non-inference* requires that the adversary should not be able to *infer* the value of a private variable based on the values of public variable. Non-inference, therefore, allows information to flow from private variables to public variables, but prohibits the adversary from learning the value of any private variable from public variables. In this dissertation, we show how non-inference can be used to preserve location privacy.

1.7 Summary of Dissertation Contributions

The main contributions of this dissertation were published in the *Proceedings of the 8th International Conference on Pervasive Computing and Communications* [63], *Proceedings of the 6th International Conference on Pervasive Computing and Communications* [68], the *Proceedings of the 7th International Workshop on Mobile Computing Systems and Applications* [67], the *Proceedings of the 3rd International Conference on Mobile and Ubiquitous Systems* [62], the *Proceedings of the 1st International Conference on Mobile and Ubiquitous Systems* [61], the *Adjunct Proceedings of the 5th International Conference on Pervasive Computing* [64, 66], the *Proceedings of the 10th International Workshop on Future Trends in Distributed Computing Systems* [41], and *Lecture Notes in Computer Science* [65].

In this dissertation, we investigate the four key issues in location-aware personal computing: (1) seamless service discovery and provisioning, (2) infrastructureless indoor localization, (3) battery management for smart phones and (3) location privacy. We provide two solutions for determining user's location indoors without requiring any extra infrastructure. The first solution uses vision algorithms to determine user location based on the images captured by the camera phone worn by the user as a pendant [67]. The second solution determines user location with room-level accuracy based on the intensity of light incident on a light sensor worn by the use [64]. These two solutions can potentially be combined together for higher accuracies.

We present a protocol for ad-hoc service discovery and provisioning [68, 61]. The services are delivered to the smart phone of the user over Bluetooth. Bluetooth SDP and 3G are used together for service discovery, interaction and payment. The protocol has been implemented and tested on Sony Ericsson P900 phones. Results show satisfactory performance in terms of service provisioning latency. We also present an application-level solution for reducing the Bluetooth discovery time in scenarios where there are multiple smart phones carrying out service discovery simultaneously.

We show how location information can aid in battery management on smart phones [66, 63]. By storing past location traces of the user, future whereabouts of the user can be

predicted. This information can be useful in estimating when the next opportunity for charging the phone will be encountered. The knowledge of when the user will charge the phone next can be instrumental in managing the limited battery lifetime on smart phones across multiple applications.

Finally, we present a novel information flow control model, called Non-inference [62], and show how it can be used to prevent unnecessary disclosure of location information to untrusted location-based services. Non-inference is enforced using static program analysis.

1.8 Contributions to the Dissertation

Peter Stern and Niket Desai implemented the applications on top of SDIPP protocol [68]. Ahmed Elgammal provided the codebase for the vision algorithms used in camera-phone-based localization [67]. Pravin Shankar and Andrew Frankel helped with the implementation and experimentation of the camera-phone-based localization system [67].

1.9 Organization of the Dissertation

In chapter 3, we describe SDIPP, a protocol for discovering and provisioning services to the smart phone. We also summarize the design of Portable Smart Messages, which are used by SDIPP for multi-hop service discovery.

In chapter 3, we describe two infrastructureless solutions for indoor localization, which use camera phone and light sensor respectively. The evaluation focuses on measuring the accuracy of the two solutions.

In chapter 4, we describe a location-aware battery management scheme for smart phones, and present experimental results obtained with real user data.

In chapter 5, we present a novel information flow control model, called Non-inference, and show how it can be used to prevent unnecessary disclosure of location information to untrusted location-based services.

We conclude in chapter 6

Chapter 2

Location-aware Service Discovery and Provisioning

Provisioning locally embedded services to the user without prior knowledge of the environment is an important aspect of location-aware personal computing, as it is not always possible to customize web services to serve as local services. Protocols are needed for seamlessly discovering these locally embedded services and provisioning them to the user. These protocols are called Service Discovery Protocols (SDP's). Several service discovery protocols (SDP's) have been proposed in the past (see Table 1.1). SDP's can be roughly classified into two categories: *client-service* model and *client-service-directory* model. In *client-service* model, services advertise themselves and clients directly query the services. In *client-service-directory* model, services cache their information on a centralized directory and clients query directories in order to discover services. While the *client-service* model requires less infrastructure and is more suitable for nomadic environments, it is not nearly as powerful as the *client-service-directory* model. There is a need for a protocol that takes a hybrid approach, such that the power of *client-service-directory* model is realized without the need for extra infrastructure.

Bluetooth is a wireless protocol designed for low-power consumption and short-range communication (1-100 meters). Its primary purpose is to connect devices such as mobile phones, printers, laptops and digital cameras. Bluetooth also has its own service discovery protocol. In Bluetooth SDP, the client continuously transmits inquiry packets and hops frequencies 3200 times per second. A service that allows itself to be discovered, regularly enters the inquiry scan state to respond to inquiry messages, hopping frequencies once in 1.28 seconds. Bluetooth SDP has been incorporated in smart phones and follows the *client-service* discovery model. Since smart phones have always-on internet connectivity through 3G/GPRS, directories can be conveniently maintained

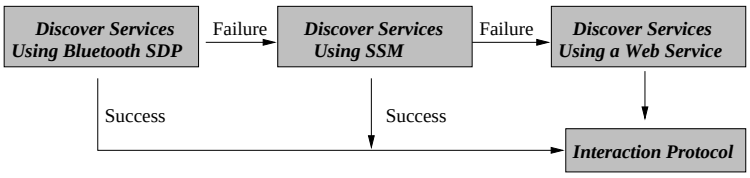

Figure 2.1: *SDIPP Discovery Model*

on the internet.

In this chapter, we present a Service Discovery, Interaction and Payment Protocol (SDIPP) [68] that follows the *client-service-discovery* model. SDIPP exploits *dual-connectivity* (Bluetooth and GPRS/3G) on smart phones to simultaneously connect with local services and web services. In the first phase, the services are discovered. In the second phase, a service is selected (based on the location of the user) and a mechanism for interacting with the service is decided. In the third and final phase, the user interacts with the service and pays for its usage (when required). In the following, we describe the basic architecture of SDIPP and the protocols involved. Portable Smart Messages [61], which is used by SDIPP for multi-hop service discovery, is also described.

2.1 Protocols

SDIPP consists of three protocols: protocol for discovering services, protocol for invoking services and protocol for paying services. In the following, we describe each of the three protocols.

2.1.1 Discovery Protocol

The discovery protocol is hierarchical in nature and builds on top of Bluetooth service discovery protocol [14] (which is *query based*), and 3G. The goal is to achieve the power of the *client-service-directory* model without the need for extra infrastructure. Bluetooth SDP provides service browsing without apriori knowledge of service characteristics. It does not include functionality for invoking services. However, it can be used in conjunction with another protocol for service invocation. The discovery protocol is

a 3-step process:

One-hop discovery: Services in the proximity (one-hop) are discovered using Bluetooth SDP. If the list of services discovered by Bluetooth SDP includes the desired service, the discovery phase is over. If it does not include the desired service, but instead lists a Service Discovery Service(SDS), the SDS is invoked to locate the desired service in a multi-hop fashion.

Multi-hop discovery: SDS would implement a mechanism for discovering services and could exploit the ad-hoc network for doing so. SDS could be implemented using any of the previously proposed service discovery protocols.

In SDIPP, multi-hop discovery is carried out using smart messages [61]. Smart Messages are user-defined distributed applications similar to mobile agents, which execute on nodes of interest defined by properties. Smart Messages migrate between nodes of interest using content-based routing, where content/property is stored in *TagSpace*. *TagSpace* is name-based memory and is composed of *tags*. A *tag* is a $(name, data)$ pair. A service is identified by a tag, which it creates on the device it runs on, for it to be discoverable by a Smart Message. The tag stores the service description. An SDS implemented using Smart Messages eliminates the need of having directories in the ad-hoc network. If a directory exists, it can be exploited but the discovery does not depend on the presence of a directory.

Web-based discovery: If no SDS is listed, the GPRS/3G connectivity on the phone is used to contact a web service that can locate the desired service. The services would have themselves registered on the web server as a way of advertising themselves and would periodically update their information on the web server. This web service would serve as a directory that lists the location of all the registered services around a particular location. A simple interval tree representation for storing location and services is used.

Since downloading data from the internet is not free of charge, directory lookup is the last step in the protocol. The personal information and preferences of the user

are stored in the *Cache* on the smart phone. The web service lists the services based on user's location, personal information and preferences. Figure 2.1 summarizes the discovery phase.

2.1.2 SDIPP Interaction Protocol

In a ubiquitous computing environment, the interaction of the client with a service is assumed to be spontaneous, which implies that the protocol for interacting with the service would have to be learnt on the fly. The interaction protocol in SDIPP is inspired by the idea of downloading the client-side code for the service on-the-fly as used in Jini [81]. Every service registers itself with a web server, which assigns it a unique id and stores the interface that can be downloaded for interacting with the service. Figure 2.2 gives a pictorial view of the interaction protocol. The protocol can be summarized as follows:

- The smart phone lists the services discovered during discovery phase. A request is sent to the desired service to send back its unique id over the Bluetooth connection.
- The service responds with its id.
- The id along with the personal information of the user stored on the smart phone is sent over to a trusted web server over the GPRS/3G connection. The personal information of the user would be used for authenticating her if the service requires that. A weaker form of authentication would be using the IMEI number of the phone, which is a 15-digit unique code that is used to identify a GSM phone.
- After an optional authentication, the web server responds with the code and data that will be used for interacting with the service. The code is a Java program that contains the protocol and interface for interacting with the service.
- Since the code is obtained from a trusted server, it is assumed to be safe code and is dispatched for execution on the phone. All further communication between the phone and the service takes place as a result of executing the downloaded code.

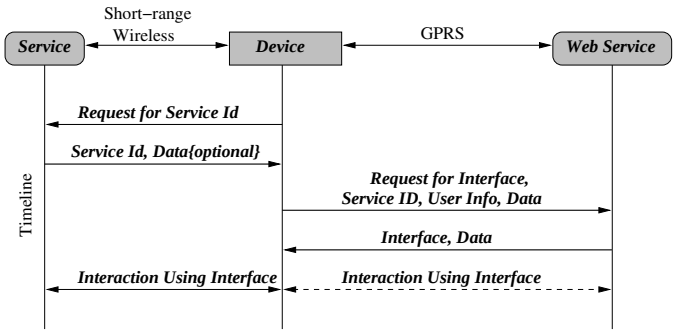

Figure 2.2: *SDIPP Interaction Protocol*

The web server(s) for storing the downloadable interface for the services may or may not be different from the web server(s) that act as service directories. Downloading of code from the internet is implemented using OTA (Over-The-Air) provisioning [9]. OTA is a method of distributing new software updates to mobile phones over the cellular network.

2.1.3 SDIPP Payment Protocol

The payment protocol is based on the electronic cash representation proposed by the Millicent protocol [35]. There are a number of existing and proposed protocols for electronic commerce such as DigiCash [5], CyberCash [4], First Virtual [6], NetBill [21] and Millicent [35]. None of these protocols has taken off due to lack of supporting infrastructure to implement them in real life.

Digital cash is normally issued by a central trusted entity (like a bank). The integrity of digital cash is guaranteed by the digital signature of the issuer, so that counterfeiting digital cash is extremely hard. However, it is trivial to duplicate the bit pattern of the digital cash to produce and spend identical (and equally authentic) cash.

Millicent proposes the idea of using accounts based on scrip and brokers to sell scrip. A piece of scrip represents an account the user has established with a vendor. At any given time, a vendor has outstanding scrip (open accounts) with the recently active users. The balance of the account is kept as the value of the scrip. When the customer makes a purchase with scrip, the cost of the purchase is deducted from the scrip's value

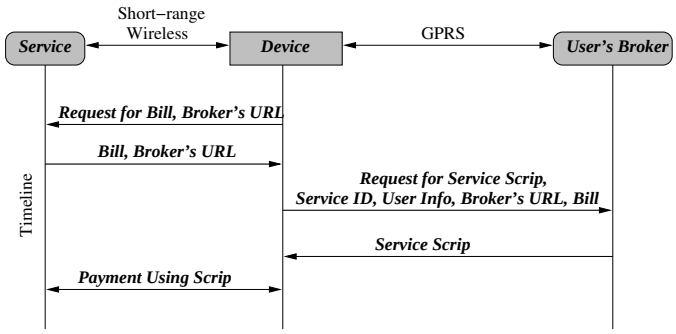

Figure 2.3: *SDIPP Payment Protocol*

and new scrip (with the new value/account balance) is returned as change. When the user has completed a series of transactions, he can "cash in" the remaining value of the scrip (close the account).

Brokers serve as accounting intermediaries between users and vendors. Customers enter into long-term relationships with brokers, in much the same way as they would enter into an agreement with a bank, credit card company, or Internet service provider. Brokers buy and sell vendor scrip as a service to users and vendors. Broker scrip serves as a common currency for customers to use when buying vendor scrip, and for vendors to give as a refund for unspent scrip.

In our model, the broker is a web service that the user already has an account with. The vendor is the service that the user wishes to use and pay for. However, Millicent model assumes that before the customer initiates transaction with the vendor, she either already has the vendor scrip or the broker scrip, which can be used to buy vendor scrip directly from the vendor. Having service/vendor scrip prior to discovering the service would imply that the user already had some knowledge about the service. This is not acceptable in ubiquitous computing scenarios where transactions are primarily spontaneous. Having broker scrip prior to discovering the service is a reasonable assumption to make, however, in order to be able to use the broker scrip, the service in question should be able to verify the authenticity of the broker scrip, which requires additional infrastructure.

Dual connectivity on phones allows the user to bypass both these assumptions because the user can be connected to the broker and the service at the same time.

Figure 2.3 illustrates the payment protocol. It can be described as composed of the following steps:

- The smart phone requests the service for its broker's URL and the bill over Bluetooth connection.
- The service responds with its broker's URL and the bill.
- The service id, broker's URL and bill amount is sent over to the user's broker over GPRS/3G along with the personal information of the user stored on the phone.
- User's broker buys service scrip from service's broker on user's behalf. The amount of scrip bought is greater than or equal to the bill amount.
- User's broker responds to the smart phone with the service scrip .
- User pays the service using the service scrip.

Brokers are assumed to be trusted services that have service providers as their clients and other brokers as their peers. Even if the broker tries to cheat, the customer and the service provider can independently check the scrip and detect broker fraud.

Service provider fraud consists of not providing service for valid scrip or deducting more amount from the scrip than is valid. If the service provider tries to cheat, the customer can detect the fraud and complain to the broker, who will take care of it.

If the customer is cheating, then the service provider's only loss is the cost of detecting the bad scrip and denying service. Every transaction requires that the customer knows the secret associated with the scrip. The protocol never sends the secret in the clear, so there is no risk due to eavesdropping. No piece of scrip can be reused, so a replay attack will fail. Each request is signed with the secret, so there is no way to intercept scrip and use the scrip to make a different request.

This payment protocol provides a security model that is well suited for profit-based services, where the service and the user need to be authenticated to each other and anonymity maintained at the same time.

2.1.4 Bootstrapping

The service registers itself on a web directory (for discovery) and a web broker (for payment). In addition, the service uploads the interface and data needed to interact with it on a web service after authentication and certification. For it to be discoverable by an SDS implemented using Smart Messages, the service creates a *tag* containing the service description.

2.1.5 SDIPP Architecture

The SDIPP architecture consists of the three protocols described above, which are implemented on top of three building blocks: *Bluetooth engine, GPRS Engine* and *Cache*. Bluetooth Engine is invoked by the protocols to discover or interact with the services in the proximity. It is a layer above the Bluetooth stack and provides a convenient Java API for accessing the Bluetooth stack. *GPRS Engine* is invoked to carry out the communication between the phone and the web services over GPRS.

Cache is persistent storage. The personal information of the user along with her preferences regarding services are stored in the cache. Personal information of the user may include name, age, address, credit card number etc. Storing personal information serves two purposes: first, it provides a way of identifying the user and authenticating her if need be; second, personal information along with preferences and location help in identifying the best suited service for the user during service discovery phase.

2.2 Portable Smart Messages

Portable Smart Messages are used by SDIPP in multi-hop service discovery. In this section, we describe the design of Portable Smart Messages which is an extension of the Smart Message(SM) [44] model.

The design of Smart Messages (SMs) has been inspired by mobile agents. A Smart Message is a user-defined application whose execution is distributed over a series of nodes using execution migration. The nodes on which SMs execute, called *nodes of interest*, are named by properties and discovered dynamically using application controlled

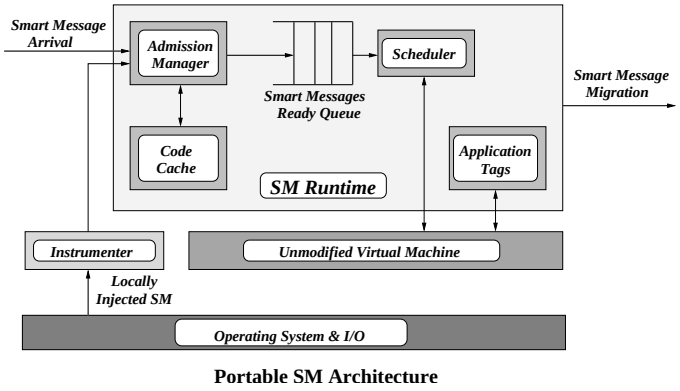

Figure 2.4: *Portable Smart Message Architecture*

routing. To move between two nodes of interest, an SM calls explicitly for execution migration, and routes itself without any underlying routing support. An SM consists of *code bricks*(e.g Java class files), *data bricks*(e.g Java objects which store data) and execution state. SMs are resilient to network failures, as they carry the code for routing themselves, and can therefore store-and-forward themselves opportunistically.

Portable SMs is an extension of the Smart Message model and has been tailored to suit the requirements of resource constrained devices (such as smart phones), which come with pre-installed Java Virtual Machine (JVM). In the design of Portable SM, special attention has been paid to portability and lightweightness. In addition to being lightweight, Portable SMs provide the functionality to support service execution, discovery, and migration in highly volatile mobile ad hoc networks.

2.2.1 Portable SM Architecture

Every participating node has to be equipped with the Portable SM middleware. The middleware is written completely in Java (using J2ME CDC and CLDC), and can be ported to the common JVMs. It consists of the following components (as shown in Figure 2.4):

Tag Space: Tag space is name-based virtual memory. It is composed of tags, which are *(name, data)* pairs. These tags are Java objects that can be created, deleted, read from, or written into by Portable SMs. Nodes are identified by properties that

are stored in tags. Also, services running on these nodes create tags for advertising themselves. Tags are, therefore, integral to content-based routing and service discovery over Portable SMs.

In addition to providing storage, tags also provide inter-SM communication and synchronization. Commonly, a blocked SM is woken up by the interpreter when the tag is written by another SM. Each time an SM blocks on a tag, its corresponding Java thread is terminated. Each time an SM is unblocked (and consequently dispatched for execution), a new Java thread is created for it.

Admission Manager: The admission manager is responsible for receiving and admitting incoming SMs over different network interfaces. Our admission manager listens on the TCP/IP socket interface (for receiving Portable SMs over 802.11b) as well as Bluetooth L2CAP interface (for receiving Portable SMs over Bluetooth). While admitting SMs into the system, the admission manager verifies the data bricks and state against certain verification policies.

Code Cache: Code cache stores frequently used code bricks. In order to implement Code cache, Java's classloader is exploited. The Java dynamic class loading mechanism is used to load a class representing a code brick. In the process, a new *Class* instance of the corresponding class is created. The classloader will not unload the class as long as there is a live reference to the *Class* instance. References to the cached classes are stored such that these classes are not unloaded by the classloader. When the caching policy chooses a class for eviction, the stored reference for that class is removed.

Scheduler: The scheduler is responsible for dispatching Portable SMs (from the ready queue) for execution on the JVM. The scheduler is implemented as a Java thread that extracts an SM from the ready queue in FIFO order, dispatches it for execution as a Java thread, and goes to sleep. When the SM completes its execution, it wakes up the scheduler using the Java's thread synchronization mechanism.

2.2.2 Portability

For implementing migratory applications or services, it is important that they be portable and transferable with minimal overhead. The original Smart Message architecture [44] was implemented by modifying Sun's Java Kilobyte virtual machine (KVM). The whole architecture was implemented inside the VM because of the need for VM support in capturing the execution state and restoring it at destination to resume the execution. This implementation, although powerful and efficient, is not portable. Since devices like Smart Phones and Smart Watches come with a pre-installed Java VM, (and most of the time users do not want to or cannot modify the system software on their devices), we designed the Portable SM middleware to execute on top of unmodified Java virtual machines.

The main issue to be solved in a pure Java implementation of a migration-based middleware is performing migration without requiring the VM to capture and restore the execution state. The execution state is located inside the VM and is not directly accessible to the external world. In order to provide migration without modifying the VM, we designed a mechanism for capturing and restoring the execution state by incorporating all the necessary operations in the SM itself. The heart of our approach lies in instrumenting the SM bytecode in such a way that the SM can save its state before migration and restore it before resumption with a minimal overhead. Using this mechanism, the state is encoded in the data bricks, and no explicit state information is shipped. Being resource and bandwidth constrained, mobile ad hoc networks impose constraints on the amount of data that can be transferred for reliable communication. One of the main goals of Portable SM middleware is to make the migration mechanism extremely lightweight and efficient. Our Java bytecode instrumentation mechanism increases the Java bytecode size by only 3% as opposed to previously proposed portable Java migration mechanisms, which increase the bytecode size by as much as 400%. The mechanism is generally applicable to any system based on execution migration of Java programs

In the following, we describe the migration mechanism, which is generally applicable

to any system based on execution migration for Java programs. To migrate a Portable SM, its code bricks, data bricks, and execution control state have to be migrated. The code bricks are Java class files, and data bricks are Java objects. Java reflection mechanism is used for loading the classes dynamically at the destination node. The Java serialization mechanism is used to marshal/unmarshal the data bricks across migrations. Since Portable SMs do not use local variables across migrations (i.e., the programmers have to include any data that they need across migrations in the data bricks), object deserialization works fine to restore the values of all objects and variables.

The main problem that needs to be solved is how to capture and restore the execution control state (i.e., located inside the VM), which consists of the instruction pointer and the method call stack. Our solution is to instrument the Portable SM bytecode in such a way that SMs can capture and restore their own runtime stack before resuming their normal execution at destination. The main idea is to label the different code sections and use a virtual instruction pointer to keep track of which code sections have already executed. At the remote site, the program is executed from the beginning and the code sections that have already executed are skipped, thereby reconstructing the call stack.

There are numerous reasons for choosing bytecode transformation [73, 79] over source code transformation [33]. First, source code transformation does not provide fine-grained control as provided by a bytecode transformation (e.g., the lack of *goto* statement in Java, the difficulty of instrumenting compound statements). Second, instrumenting a loop in source code requires the loop to be unfolded in order to preserve correct execution semantics. Third, instrumenting the source code causes the corresponding bytecode to blow up, and therefore, incurs heavy overheads.

Definitions

We use the term *critical method* to refer to any method that can directly or indirectly invoke *sys_migrate* or *blockSM*. These are the only two methods that can lead to migration and hence capturing and restoring of the execution control state. Therefore, only critical methods need to be instrumented. Since a migration (or block) happens at the

end of a method call chain, the instrumenter has to detect all the methods from which *sys_migrate* (or *blockSM*) is statically reachable in order to recognize critical methods. To simplify the exposition throughout this section, we will refer only to migration.

The bytecode instrumenter adds an integer array $ip[length]$ to every class, where *length* is the number of methods in that class. An element $ip[i]$ is used as a pseudo instruction pointer for the *ith* method. The code of a critical method is divided into *code regions* separated by critical method invocations. A critical method invocation marks the end of a code region and the beginning of another new code region.

Approach

The value of $ip[i]$ is incremented only before a critical method invocation, and hence, serves as a pointer to the boundaries between code regions. At the time of resumption, the value of $ip[i]$ also serves as a pointer to the last statement executed inside the *ith* method of the class.

The last statement executed inside a critical method before a migration is always a critical method invocation (i.e., either directly a *sys_migrate* call or a chain of method invocations that ends with a *sys_migrate*). This is the reason why incrementing the value of $ip[i]$ only before *critical* method invocations is sufficient. The value of $ip[i]$ can be used during resumption to locate the last method invocation made from method i before migration. Since every object has a unique *ip* associated with it, *ip* is carried over as a part of data bricks and restored during deserialization.

During resumption, each SM starts its execution from the beginning of the *run()* method of the main class (i.e., Portable SMs execute as Java threads). The instrumenter introduces a *switch* statement at the beginning of every critical method to redirect the instruction pointer, based on the value of $ip[i]$, to the last statement executed before migration. Hence, the code already executed is skipped. For every method other than the one that directly invoked *sys_migrate*, this will result in an invocation of the method that was adjacent to this method in the runtime stack before migration. As a consequence, the runtime stack is recreated. The $ip[i]$ of the method that directly invoked *sys_migrate* serves as a pointer to the statement immediately following the

```
class A{
   void <init>{
     ...
   }

   void Method1(){
     int j = 0;
     int i = 0;
     System.out.println("hello");
     ...
     Method2();
     ...
     Method3();
     ...
     Method4();
     ...
     Migration.sys_migrate();
     ...
     TagSpace.blockSM();
     ...
   }
}
```

Figure 2.5: *Portable SM Pseudo-code before Instrumentation*

sys_migrate call.

An SM is said to be in *resumption mode* when it is recreating the runtime stack. To differentiate between *resumption mode* and *normal execution*, the instrumenter adds a global flag: *resumption*. This flag is important for preserving the correct execution semantics. Its purpose is to activate or deactivate the *switch* statement introduced by the instrumenter at the beginning of each critical method depending on whether the SM is undergoing normal execution or is in resumption mode. If the SM is resuming, it is necessary to execute the *switch* statement in order to skip the already executed code. If the SM is undergoing normal execution, it is necessary to ignore the value of $ip[i]$ to ensure that a method invocation is not influenced by this value (i.e., $ip[i]$ might be non-zero due to an earlier invocation of the same method). The *resumption* flag of the SM is set by the system before the SM is migrated and reset by the SM itself once the SM has reconstructed the method call stack, at which point *normal execution* of the SM begins. To achieve this, the *resumption* flag is reset after every statement containing a call to *sys_migrate*.

```
class A{
   public int[] ip;
   void <init>{
      ip[] = new int[5];
      ...
   }
   void Method1(){
     int j = 0;
     int i = 0;
     if(SM.resumption == true){
        switch(ip[1]){
           case 0: goto label 0;
           case 1: goto label 1;
           case 2: goto label 2;
           case 3: goto label 3;
           case 4: goto label 4;
        }
     }else{
         ip[1] = 0;
     }
     label 0 : System.out.println("Hello");
               ...
               ip[1]++;
     label 1 : Method2();
               ...
               Method3();
               ...
               ip[1]++;
     label 2:  Method4();
               ...
               ip[1]++;
               Migration.sys_migrate();
     label 3:  SM.resumption = false;
               ...
               ip[1]++;
               TagSpace.blockSM();
     label 4:  SM.resumption = false;
               ...
   }
}
```

Figure 2.6: *Portable SM Pseudo-code after Instrumentation*

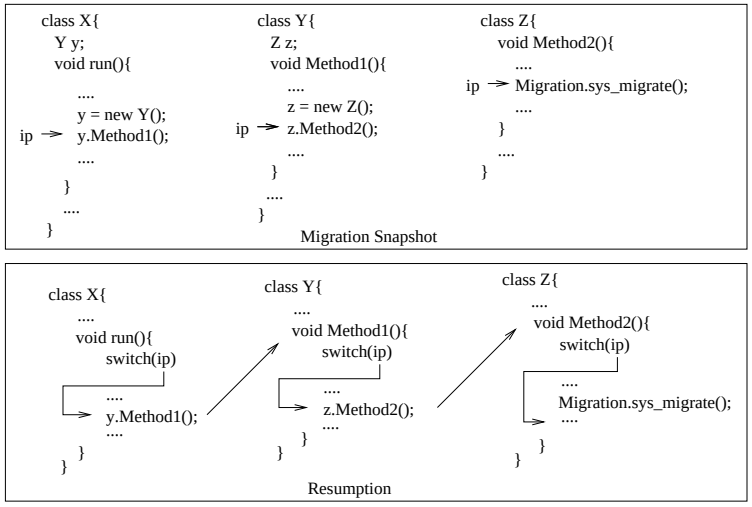

Figure 2.7: *Example of Resuming SM Execution after Migration*

Example

Fig. 2.5 and 2.6 illustrate the transformation done by the bytecode instrumenter. Although the transformation is done on the bytecode, for the sake of simplicity, we show a higher level transformation on the corresponding Java pseudo-code. In the example, $classA$ has four methods, excluding the constructor. Let us assume that $Method1$, $Method2$, and $Method4$ are critical methods (i.e., they can directly or indirectly invoke $sys\_migrate$ or $blockSM$), while $Method3$ is not a critical method. We present the bytecode instrumentation only for $Method1$, but similar transformations take place on the other critical methods ($Method2$ and $Method4$ in this case) as well. As $Method3$ according to our assumption is not a critical method, it is not instrumented.

Since $classA$ has five methods including the constructor, ip is declared as an array of length five. We initialize this array in the $<init>$ method which is internal to the bytecode and is invoked every time a new object of the class is created. Given that $Method1$ has four invocations to critical methods (two indirect, and two direct), its code is divided into five code regions labeled from 0 to 4. The value of $ip[1]$ is incremented before every invocation to a critical method. For instance, $ip[1]$ is incremented before an invocation to $Method2$, but not before an invocation to $Method3$ which is not a critical method.

Table 2.1: *Performance Evaluation of SDIPP*

Operation	Average Time of Completion
Bluetooth Service Discovery	22.5 sec
Ad-hoc Service Discovery	2 sec × No. of Hops
Web directory lookup	2.5 sec
Interaction Protocol(Lower Bound)	3 sec
Payment Protocol	6 sec

Suppose $Method1$ had called *sys_migrate* before migration, the value of $ip[1]$ would be 3. When the SM resumes execution at the destination node and enters $Method1$, the instruction pointer would be redirected to *label 3* by virtue of the *switch* statement; from this point on, *normal execution* of the SM begins. If on the other hand $Method4$ had called *sys_migrate*, then the value of $ip[1]$ would be 2. When the SM enters $Method1$ after resuming at the destination node, the instruction pointer would be redirected to *label 2* which contains a call to $Method4$, thereby skipping the already executed code in $Method1$ and recreating the runtime stack.

This example also shows how the *resumption* flag is used. If the flag is set to *false*, the execution of the methods starts from the beginning. Otherwise, it starts with the code region pointed to by $ip[1]$. As soon as the SM recreates the stack, the *resumption* flag is reset by the SM itself. This ensures that any future invocation to $Method1$ or any other critical method will not be affected by the value of *ip*. Note that the *resumption* flag is local to an SM, but global to all the classes that constitute that SM.

Fig. 2.7 briefly demonstrates the working of our instrumentation scheme. The upper part of the figure gives a pictorial view of *ip* in three critical methods at the time of migration. The arrows in the lower part of the figure show the control flow of the SM from the time of execution resumption at the destination until the method stack is recreated.

2.3 Implementation and Evaluation

The SDIPP protocol was implemented and tested on Sony Ericsson P900 smart phones. MIDP and JSR-82 (Java Bluetooth API) were used to implement the architecture. Table 2.1 shows the time of completion for the different phases of the SDIPP protocol. The time of completion of the Interaction Protocol depends on the size of the code downloaded from the Internet. The lower bound is determined by the size of the *jad* file of the corresponding code which is typically 250 Bytes. The time of completion of the ad-hoc service discovery over Smart Messages depends on the number of nodes (hops) involved.

Device and service discovery together on an average takes around 22.5 seconds to complete on P900 phones. We implemented and tested a few applications on top of SDIPP. For details refer to [68].

Portable SM architecture was ported and tested on HP iPAQs and Sony Ericsson P800/P900 phones. Personal Java and C++ (connected through JNI) were used to port SMs on phones. Table 2.2 compares the cost of Portable SM execution (including migration) on Sony Ericsson phones with that on HP iPAQs. The results indicate that, for establishing a Bluetooth connection, it takes on an average a constant of 1 second, and the round-trip time varies from 300 ms to 1600 ms (excluding the cost of establishing a Bluetooth connection) as data brick size is varied from 1KB to 16KB. For all practical purposes, this is good performance. The performance on iPAQs is much better compared to that on Sony Ericsson phones, which is expected because iPAQs have more computation power than smart phones and 802.11b offers a much higher bandwidth than Bluetooth.

2.3.1 Case Study

In this section, we briefly describe an application that was developed on top of the SDIPP protocol, for better understanding of how the protocol works. The application allows a user to open a Bluetooth-enabled door using their smart phone.

Table 2.2: *Effect of Data Brick Size on Single-Hop Portable SM Round-Trip Time*

Size(Bytes)	Round-Trip Time(ms)	
	HP iPAQ	Sony Ericsson P800/P900
1044	150	1450
2088	177	1600
4056	196	1790
8010	234	2120
16010	301	2630

The Bluetooth-device attached to the door runs a service that can be used to authenticate the user and receive command for opening the door. The user's phone is assumed to be equipped with the SDIPP protocol. In the first phase, the services in the vicinity are discovered using the discovery protocol. At the end of the discovery phase, the room numbers are listed on the phone. The user selects the room he wants to enter. This starts the second phase, in which the interface for interacting with the service and the relevant data (which in this case is the digital key) are downloaded from a web server over 3G over an encrypted channel. During this phase, the personal information of the user is sent to the web server. The web server responds with the interface and key after authenticating the user based on the personal information. The downloaded interface is then automatically dispatched for execution, which allows the phone to interact with the door service. During this phase, the key is communicated to the door service over Bluetooth. The door service authenticates the key and unlocks the door, allowing the user to enter.

We demonstrated this application on the doors of the Computer Science Department at Rutgers University. We emulated the Bluetooth enabled door device using a smart phone which runs the door service. The user's phone only interacts with the door service and the web server. The user has to be registered with the web server which maintains an access control list and key database for every digital door. The door service communicates with a backend computer that controls the doors. We built our own control circuit driven by the backend computer's serial port that controls the power supply to the door lock. On receiving a signal from the door device, the computer sets the Data Terminal Ready Bit to 1 which switches on a 24V power supply to the door

lock thereby opening it. Excluding discovery, the whole procedure takes around 5.5 sec to complete.

2.3.2 Bluetooth Inquiry Clash

During the process of implementing SDIPP, we found that Bluetooth discovery can be quite time consuming when multiple devices are simultaneously in the inquiry mode. We devised an application level solution to minimize Bluetooth discovery time. In this section, we describe the solution.

When a Bluetooth device is in the inquiry mode, it continuously transmits inquiry packets and hops frequencies 3200 times per second. A device that allows itself to be discovered, regularly enters the inquiry scan state to respond to inquiry messages, hopping frequencies once every 1.28 seconds. In order to discover all devices, the device must spend at least 10.24 seconds in the inquiry mode [17] [14]. Once discovery has completed, the device behaves like other devices, entering the inquiry scan state periodically. When device discovery is followed by service discovery the minimum time to be spent in the inquiry mode is even larger.

On an average a P900 phone takes 22.5 seconds to discover all devices and services. Inquiry is a process that inside most basebands takes up all the radio bandwidth. This means that a phone, when in the inquiry mode, cannot use the radio for anything else except actions like *local friendly name retrieval* or *local parameter retrieval*, which implies that the services on the phone become indiscoverable while it is in the discovery mode.

We can envision applications being deployed in the future where Bluetooth devices are continuously trying to discover other Bluetooth devices/services around them. For example, in the previously described case study, the door devices could poll instead of the phones. Another example is that of the WAP-push based advertising system described in [17].

This problem is similar in nature to that of nodes connected on an ethernet, each trying to send data periodically, but unlike ethernet which is a static system, this is more dynamic in nature as new devices are encountered and old ones disappear due to

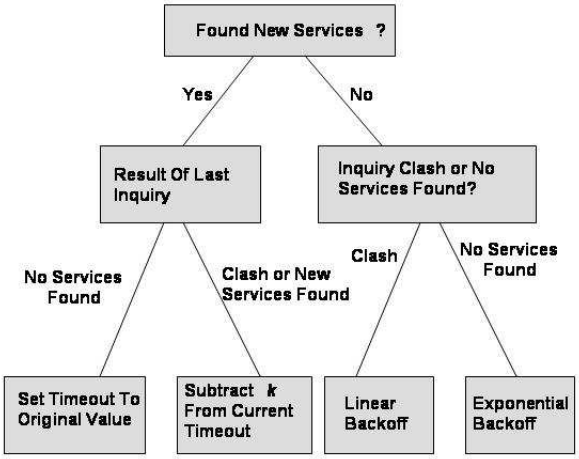

Figure 2.8: Policy for Timeout For Minimizing Bluetooth Inquiry Clash

Table 2.3: Comparison of *Random* With Our Policy for Minimizing Bluetooth Inquiry Clash

Number of Phones	Discovery Completion Time	
	Random	Policy
3	10 min	5 min
4	18 min	12 min

mobility. The solution to this problem (at the application layer) lies in choosing the optimal timeout value t between two consecutive polls. A very small t would result in large number of clashes while a very large t would result in inefficiency.

If the timeout t is constant, two phones running the same application and hence polling after every t seconds will never be able to discover each other. A better choice would be to choose t randomly between a certain interval $[0, r]$. We call this strategy *Random*. Let c denote the time it takes for discovery to complete, during which time the services on the device become indiscoverable. Every device is discoverable with probability $1 - c/r$.

Let X_i be a random variable denoting the time instant when device i started discovery, $0 \leq X_i \leq r$. The probability p of two devices i and j being able to discover

each other in the time interval $[0, r]$ is given by

$$P(\mid X_i - X_j \mid > c) \tag{2.1}$$

which comes to $(1 - c/r)^2$. Let $Y_{ij} (i \neq j, Y_{ij} = Y_{ji})$ denote the number of attempts needed for i and j to discover each other. Y_{ij} follows a geometric probability distribution. Let Y be a random variable denoting the number of attempts after which n devices would have all discovered each other.

$$Y = \sum_{i,\, j>i} Y_{ij} \tag{2.2}$$

and therefore follows a negative binomial distribution. Expected value of Y is given by

$$E(Y) = \frac{\binom{n}{2}}{p} \tag{2.3}$$

where p is $P(\mid X_i - X_j \mid > c) = (1 - c/r)^2$

Let T be the random variable that denotes the time after which n devices would have all discovered each other.

$$T = rY \tag{2.4}$$

$$E(T) = rE(Y) = \frac{r\binom{n}{2}}{p} \tag{2.5}$$

In our case c is 22.5 seconds. We chose r to be $2c$ which is 45 seconds, for best results. Substituting the values, we obtain the expected time for $n = 3$ devices to be 9 minutes and for $n = 4$, it is 18 minutes. This matched very well with the experiments. However, this is far away from the ideal, which for 3 devices is $3c$ which is 67.5 seconds (1 minute 7.5 seconds) and for 4 devices, 90 seconds (1 minute 30 seconds). However, the ideal is achievable only with an oracle aware of all devices, that instructs the devices when to start discovery.

After careful analysis and experimentation, we came up with a policy for varying t dynamically that seems to do much better than *Random*. The policy was motivated by the backoff algorithms used to minimize clashes in an ethernet. In addition, it also takes into account the dynamic nature of the problem by keeping track of events that happened during the last discovery attempt. The user may suddenly enter a region where there are many new services available in which case he should poll more often,

however, if there are no new services found or if a clash happens, then the value of t should be increased. Figure 2.8 shows the policy. For best results, original(initial) value of timeout t was chosen to be r (45 seconds) and k which is a constant, was chosen to be 15 seconds. Table 4.1 shows how this policy compares with *Random*. We believe there is room for improvement. Better results could be obtained by using machine learning techniques to learn the policy for varying t.

2.4 Discussion

SDIPP assumes that services are SDIPP-aware. This is a limitation of our approach, which is common with other service discovery protocols. Making services oblivious to the discovery protocol is a challenging problem. This can be alleviated to some extent with the use of ontologies, however, the solution is not clear.

Bluetooth has its own limitations, some of which have been pointed out in this chapter, such as long discovery time. SDIPP builds on top of Bluetooth and the limitations of Bluetooth, therefore, affect SDIPP's performance.

It is important to build lightweight service discovery protocols that can be widely used. A widely used service discovery protocol enables easy deployment of services. Since mobile phones are ubiquitous, it is wise to build service discovery protocols that are phone-centric.

2.5 Summary

In this section, we described the design and implementation of the Service Discovery, Interaction and Payment Protocol (SDIPP). We also presented a case study and a solution for the Bluetooth Inquiry Clash problem. The evaluation results indicate satisfactory performance. The design of Portable Smart Messages (which aid in service discovery and execution) was also presented. SDIPP relies on user location for web-based service discovery. Location can be easily determined outdoors with the help of GPS, which uses the existing satellite infrastructure. There is no such globally accepted solution for determining user location indoors. The next chapter describes how user

location can be determined indoors without extra infrastructure to be installed in the environment.

Chapter 3

Determining User Location Indoors

In order to enable location-aware personal computing, continuous location updates are needed both outdoors and indoors (as described in , SDIPP also requires user location for web-based service discovery). Global Positioning System (GPS) is a globally accepted solution for determining user's location outdoors. It uses the existing satellite infrastructure for determining user location. The user is required to carry a GPS receiver, which calculates its position by measuring and triangulating the distance between itself and three or more GPS satellites. No such globally accepted solution for indoor localization exists. This is due to the lack of a uniform infrastructure for indoor localization, unlike GPS, which utilizes the existing satellite network. Each of the previously proposed solutions for indoor localization relies on some infrastructure for sensing location, as shown in Table 1.2. This makes deployment hard and costly.

A practical solution for indoor localization should be infrastructureless, precise and robust. As has been suggested before [48], one feasible way to accomplish this is to create a fusion system that combines information from an array of body-worn sensors, each of which senses location information using existing infrastructure. While individual sensors may not be precise and may only work under certain conditions, the fusion system as a whole should be quite robust. In this chapter, we demonstrate the use of two sensors for indoor localization: phone camera and light sensor. We show that each of them is capable of sensing location with room-level precision.

3.1 Localization Approach

In this section we describe the main ideas behind the two localization solutions.

3.1.1 Location Determination with Camera Phone

In camera-phone based localization, the camera-phone is assumed to be worn by the user as a pendant(Figure 3.1). The camera captures images periodically and sends them to a web server over 3G/GPRS. The web server has a database of images with their corresponding location. Upon receiving an image, the web server compares it with stored images, and based on the match, estimates user's location. This is accomplished by using off-the-shelf image matching algorithms and tailoring them for this purpose. Images correspond to "corners". Figure 3.3 shows two low-resolution images with different locations corresponding to the same corner. First image was taken with camera in the south-east quadrant of the room. Second image was taken with camera in the north-east quadrant of the room (as shown in Figure 3.2). The localization accuracy is further improved by taking user trajectory into account.

This solution does not rely on any external infrastructure. Neither custom hardware, nor wireless access points are required. Physical objects do not have to be "tagged" and users do not have to carry any special device. The only cost involved is that of building an image database, which is common with other localization solutions [82, 39, 22, 60, 48, 75, 27, 56]. User orientation is also determined along with location.

3.1.2 Location Determination with Light Sensor

In light sensor based localization, the light sensor is assumed to be worn by the user either as a pendant or top of a hat (or shoulder). Light is the term used for electromagnetic radiation of frequencies in the band 4×10^{14}Hz to 8×10^{14}Hz. *Luminous flux (F)* is the measure of effectiveness of light in producing visual sensation and is directly proportional to the brightness of the light source. The *luminous intensity (I)* of a point-source is a measure of the luminous flux (F) it produces per unit solid angle (Ω), and is defined as the ratio $dF/d\Omega$. Finally, *illumination (E)* is a measure of the luminous flux (F) falling per unit area (A) of a surface (e.g a light sensor) and is defined as the ratio dF/dA. Illumination is measured in *lux*, and is related to intensity I by the expression: $E = Icos\varphi/r^2$, where r is the distance of the surface from the light-source

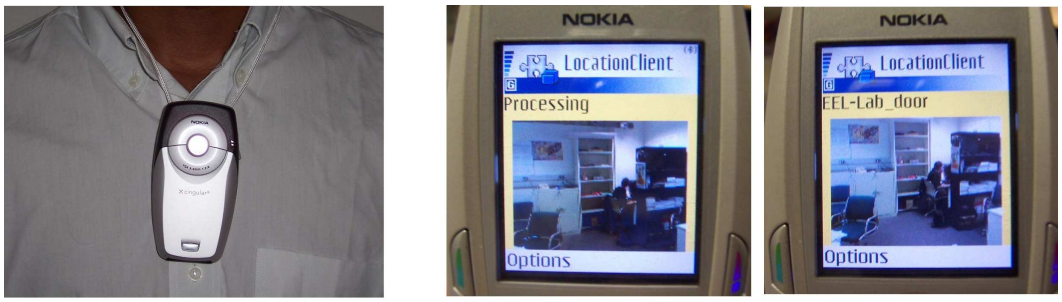

Figure 3.1: *Left: Phone as Pendant, Right: Snapshot of Client*

and φ is the angle the normal to the surface makes with the incident radiation. While intensity is a property of the light source, illumination depends on the distance of the surface from the light source and its orientation.

The light sensor measures illumination, which is intuitively the amount of light energy incident per unit area of the light sensor. Illumination is measured in *lux* and is directly proportional to the intensity of the light source. The key insight is that the distribution of light intensity inside a room is unique under static lighting conditions, which can be used to fingerprint a room. This solution also does not require any extra infrastructure to be installed in the environment. Existing light sources are used for localization. In addition, it only takes around a minute to collect training data for a room, which is very efficient.

3.2 Data Collection

Both camera-phone and light sensor require data to be collected for every room that is used to train the localization algorithms. In the case of camera-phone, it is necessary to build a database of images for the different rooms in the building. In order to minimize the overhead of data collection, we use a Java client that runs on the phone and sends images to the web server, as the database creator walks around. The web server extracts features from the received images on the fly and stores the features in a database. The images/features in the database are manually tagged with location afterwards. The process of tagging images with location can be partially automated by

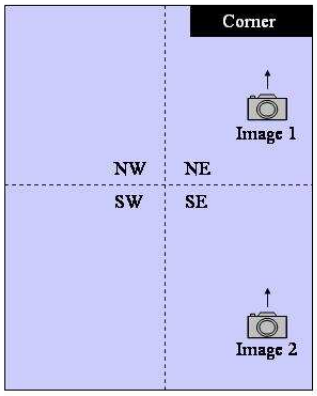

Figure 3.2: *Room Layout*

Figure 3.3: *Two Low-resolution Images of the Same Corner*

using a speech recognition interface on the phone, so that the database creator can tag images by announcing location while pictures are taken.

During image database creation, multiple images of the same *corner* are taken to accommodate for different heights and angles. On an average ten images per corner are stored. The success of the system depends on the amount of effort put into database creation. More images per corner increases the chance of success. The number of images in the database is also determined by the desired frequency of location updates. High update frequency requires high coverage, which implies a dense image database to accommodate for small changes in the location of the user.

Figure 3.4: *Left: Light Sensor on a Hat. Right: Light Sensor as a Pendant*

Figure 3.5: *Apparatus Used for Data Collection for Light-based Localization*

In the case of light sensor, it is necessary to build a database of light intensity readings for the different rooms in the building. For that, the database creator walks around in every room at slow constant speed wearing the light sensor. The light sensor is connected to a light meter (Figure 3.5), which records the readings of the light sensor in *lux*. The light meter is in turn connected to a laptop, which logs the data. Using the laptop is optional; the light meter itself is capable of logging the data. The probe-frequency can be varied– we chose 0.5 second for our experiments. The training data is processed to obtain fingerprints for every room.

3.3 Localization Algorithms

In this section, we describe the algorithms used for determining location using each of the two sensors.

3.3.1 Location Determination with Camera-phone

When the web server receives a query image, it compares it with the images stored in the database. Every image that matches with the query image is assigned a weight that reflects the degree of similarity between the two images. By image comparison, we imply feature comparison. Three off-the-shelf algorithms are used for image comparison: Color Histograms [76, 54, 80], Wavelet Decomposition [42] and Shape Matching [45].

Color Histogram: A color histogram of an image is a three dimensional distribution that reflects the probability of existence of certain color in the image. For example, if the image is represented in the RGB color space, then each pixel has three values (r,g,b): red value, green value and blue value. In this case, to build the color histogram of an image, each of the axes in the RGB color space is divided into certain number of bins (this operation called quantization). For example if the red values is quantized into k bins and the green into l bins and the blue into m bins, this gives us a total of $k \times l \times m$ bins. Each pixel in the image is mapped into its corresponding bin. The number of pixels that are mapped into each bin are counted and that give us the color histogram of the image. The color histogram can be represented as a N-dimension vector where N=$k \times l \times m$.

In our system we implemented two kind of color histograms: RGB color histogram and HSV color histogram. The RGB color histogram is built directly from the images while for the HSV color histogram, each pixel has to be transformed from RGB into HSV color histogram first. The RGB color space is not perceptually uniform. Also RGB histograms suffer from sensitivity to lighting conditions, i.e., the same image may have different color histogram under different lighting conditions. The HSV color histogram has the advantage of separating the color information from the light information that make it possible to have more bins for the colors and fewer bins for the lightness and saturation information. That makes HSV histograms more robust under different lightening conditions. We use HSV histograms in our system. There are numerous metrics for measuring histogram similarity. We use the L1 metric, also known as the

city block distance:

$$D(I, J) = \sum_{i=1}^{N} \mid I_i - J_i \mid$$

where D represents the distance (i.e dissimilarity) between two images I and J.

Shape Matching: The basic idea behind shape matching is to detect the edges in the images and to build an edge map representing the location of edges in the image. The similarity is then measured using correlation between windows in both images.

In our implementation, we used *Sobel operator* to detect the edges in the image. The image is then divided into $M \times M$ blocks. $M \times M$ edge map (2 dimension array) is constructed by setting edge map entry to 1 if its corresponding block contains certain percentage of edges and 0 otherwise. The similarity between two edge maps is performed by correlating each window from one image with corresponding set of windows in the other image and calculating edge matching score. That is, for each window centered at location (i, j) in the query image edge map, a set of windows centered at the neighbors of (i, j) is considered in the database image edge map and for each window from the query edge map and each window in the database edge map, a score is calculated by counting the edge-edge matching. The overall score function between a query edge map A and an edge map B stored in the database is S(A,B):

$$S(A, B) = \sum_{i=1}^{i=M} \sum_{j=1}^{j=M} F(A_{ij}, B)$$

where

$$F(A_{ij}, B) = \sum_{u=i-K/2}^{u=i+K/2} \sum_{v=j-K/2}^{v=j+K/2} s(A_{ij}, B_{uv})$$

where K is the size of the neighborhood tested around pixel (i, j) of the edge map and the function $s(A_{ij}, B_{uv})$ calculate the score of matching a window of size $w \times w$ centered at pixel (i, j) in A with a window in B centered at (u, v).

Wavelet Decomposition: This technique is an implementation of the approach used for content-based image query based on wavelet decomposition. This technique performs wavelet decomposition of each of the color channels (red, green, blue) of the

image based on Harr bases to represent the image with some coefficients. Then the coefficients with largest magnitude are used and the rest are truncated. The coefficients are then quantized to +1 or -1 values in order to speed the search and reduce the storage. The similarity of a query image with an image from the database is based on the weighted distance between the coefficients representing the query image and the coefficient representing each image in the database. No dimension reduction is used in this technique, instead inverted file is used to store for each coefficient which images have positive or negative values in each color channel.

Each algorithm described above assigns a weight to the image. The total weight of an image is calculated as a linear combination of the weights assigned by each algorithm. In the current implementation of the system, color histograms is assigned a weight of 1 and the other two algorithms are assigned a weight of 0. As a result, only color histograms is used for comparison. Once the weight of the images in the database with respect to the query image are known, the following methods can be used for location determination:

Naive Approach: In this approach, the images in the database are organized in a flat manner. The location of the user is the location of the image that matches the query image with maximum weight. In other words, the location of the user is the one that maximizes the probability of seeing the query image.

Hierarchical Approach: In this approach, the images in the database are organized hierarchically. The images corresponding to a floor are grouped together, the images corresponding to a room are grouped together and so on. When the system determines that the user has entered a particular room, subsequent searches are performed only on the images of that room, until the system discovers that the user has exited the room. There are two advantages of this approach: (1) the search gets localized and hence response time decreases, and (2) the probability of error decreases, because the system has fewer images to confuse the query image with. The disadvantage of this approach is that if the system incorrectly determines the room the user is in, subsequent searches

would get affected and produce wrong results.

History-based Approach: In this approach, the web server keeps track of the trajectory of the user. Based on the past locations of the user, a better estimate of the current location can be derived. In other words, the location of the user is determined not from a single query image, but multiple query images received over a certain period of time. When the server receives a query image, it looks at the last *n-1* query images. The current location of the user is the one that maximizes the probability of *seeing* the n query images in the shortest period of time.

To implement this, a simple shortest-path/nearest-neighbours approach is used. Upon receiving the *nth* query image, the web server carries out a search (image comparison) based on the last n query images. Let k_i denote the set of top 10 images that match with the *ith* query image, and let $location(k_i^j)$ denote the location of the *jth* image in that set. The weighted euclidean distance between the *jth* image of set k_i and the *mth* image of set k_{i+1} is given by :

$$d_i^{j,m} = w * |location(k_i^j) - location(k_{i+1}^m)|.$$

where w is inversely proportional to the weight of image k_i^j.

$Path$ is the set of images (one corresponding to each query image) such that the sum of the weighted euclidean distances between these images is the minimum among all possible combinations. In other words, these images define the shortest possible path (scaled by the weight of images) that the user could have traversed, and also the most likely.

$$Path = \{\forall_{i=1}^{n} k_i^j | D_i^{j,k} = d_i^{j,k} + D_{i+1}^*, D_i^* = min_{j,k} D_i^{j,k}\}$$

The location of the image in set k_n that belongs to the set $Path$ is returned as the current location of the user:

$current\_location = location(k_n \cap Path)$.

We use the *sliding-window* approach for keeping track of the history. The window of n images to be considered for calculating the shortest-path slides by 1 with every new query image. The intuition behind using the shortest-path/nearest-neighbours approach is the following: assuming that the user does not abruptly increase her walking/running speed above a certain limit, the current location of the user should not be too far away from her past locations. We use weighted distance in order to give higher priority to images with higher weight (i.e images that match better with the query image). This is the reason for using a multiplicand w, which is inversely proportional to the weight of the image. In effect, instead of estimating the location of the user based solely on the similarity with the query image (as in the Naive and Hierarchical approaches), the location is estimated by first estimating the most likely path that the user could have traversed. The most likely path is the one that maximizes the probability of "seeing" the last n query images. Every time a new query image is received, the top 10 matches corresponding to each of the last n query images are used, instead of directly using the last n estimated locations. This is done in order to account for possible error in the estimation of past locations.

Figure 3.6 shows the various components of the system. Client-side components were implemented in Java using MMAPI [2] which is supported on some Symbian OS phones. Nokia 6620 mobile phones were used for experiments. All server side components were implemented in C++ for performance reasons.

3.3.2 Location Determination with Light Sensor

The amount of light energy incident on a light sensor depends on the intensity of the light sources in the room as well as reflection from walls and furniture inside the room. While light illumination at a particular location inside a room may be the same as that at another location in a different room, the distribution of light illumination values in rooms are quite distinct. The problem of identifying which room the user is in based on a set of light illumination values, can be formulated as a classification problem for which

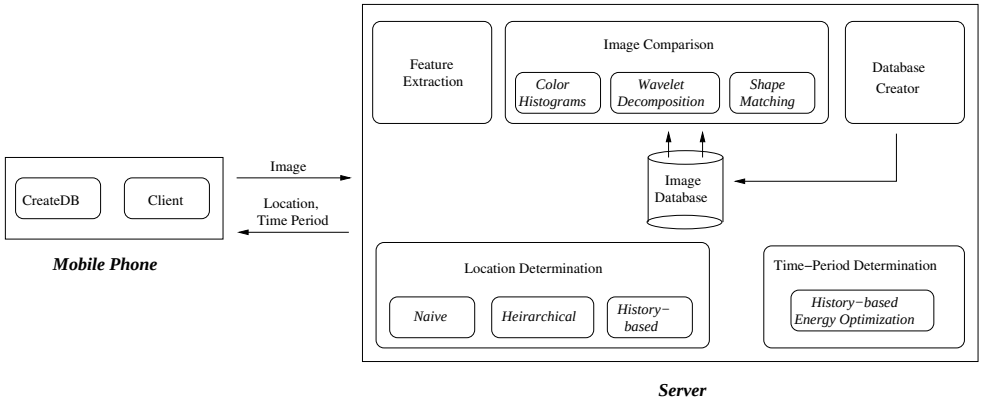

Figure 3.6: *System Architecture for Camera-phone Based Localization*

fingerprints of rooms need to be created. In what follows, we describe two algorithms for fingerprinting and identifying rooms.

Bayesian fingerprinting: In Bayesian fingerprinting, the light illumination values of a room collected during the training phase are discretised into a number of intervals. A histogram is then obtained by counting the number of points that lie in each interval. Each point corresponds to the value of light illumination at a certain location in the room. Therefore, the height of an interval is directly proportional to the number of locations in the room with light illumination value in that interval. In order to create such a histogram, the data collector must spend equal amount of time at every location in the room during the data collection phase. Figure 3.7 shows the Bayesian fingerprints (or histograms) for four rooms. It is evident from the figure that the fingerprints are quite distinct. This algorithm does not take into account the spatial distribution of light illumination in the room.

Bayesian localization: During the testing phase, when a new set of light illumination values s is obtained, Bayesian localization finds the room R that maximizes the probability that the points in set s have been sampled from room R, denoted by $\arg\max_R P(R|s)$. Applying Bayes rule, we get:
$\arg\max_R P(R|s) = \arg\max_R P(s|R)$, assuming $P(R_1) = P(R_2) .. = P(R_n)$.

We know that $\arg\max_R P(s|R) = \arg\max_R \Pi_i \{P(a_i|R)|a_i \in s\}$. Probability of sampling a point from a room is directly proportional to the height of the histogram

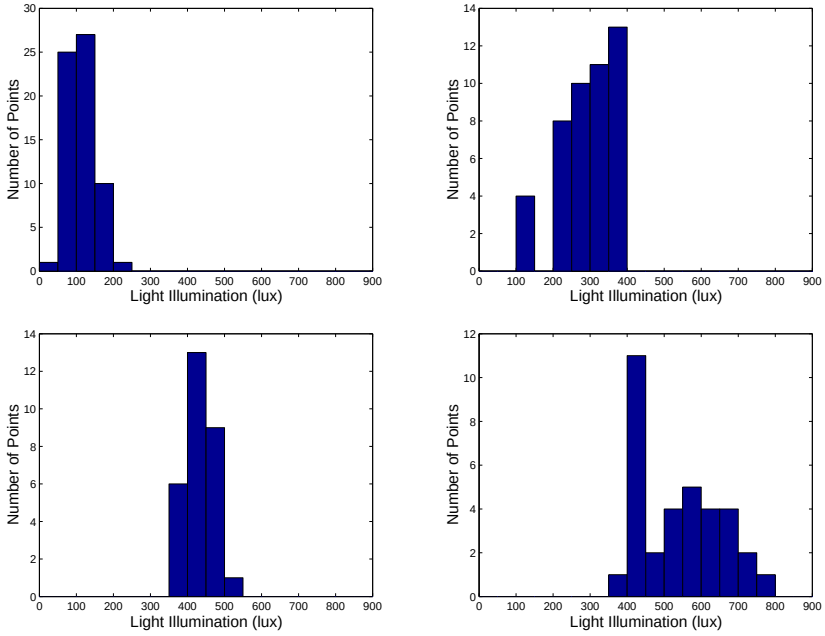

Figure 3.7: *Bayesian Fingerprints (Histograms) for Four Rooms*

interval in which the point falls: $P(a_i|R) = h_R(a_i)/\sum_i h_R(a_i)$, where h_R denotes the histogram for room R, and $h_R(a_i)$ denotes the height of the interval in which a_i falls. From here we obtain,

$$\arg\max_R P(R|s) = \arg\max_R \prod_i (h_R(a_i)/\sum_i h_R(a_i)) \tag{3.1}$$

Intuitively, the probability that a set of points s has been sampled from room R is directly proportional to the product of the heights of the histogram intervals in which the points in s fall.

Range-max fingerprinting: In Range-max fingerprinting, only the minimum and maximum values of light illumination obtained for a room during the training phase are retained. The fingerprint of a room is given by the tuple $\{min, max\}$, which denotes the range of light illumination values in that room. Figure 3.8 shows the Range-max fingerprints for six rooms.

Range-max localization: During the testing phase, when a new set of light illumination values s is obtained, the Range-max localization algorithm finds the room R that minimizes the quantity $abs(max(R) - max(s))$ and for which $min(R) \leq min(s) \leq$

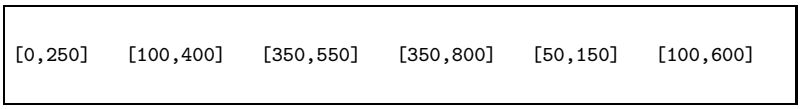

Figure 3.8: *Range-max Fingerprints for Six Rooms*

$max(s) \leq max(R)$. Quantities $min(R)$, $max(R)$ are the minimum and maximum values of light illumination for room R as obtained from the fingerprint. Quantities $min(s)$, $max(s)$ are the minimum and maximum values of light illumination in set s. Intuitively, this algorithm assumes that every room has a unique *max* value, and that the range and the *max* value together can be used to identify a room.

3.4 Evaluation

In this section, we describe the experiments that were carried out to evaluate the two localization systems and present the results.

3.4.1 Experimental Results for Camera-phone Based Localization

A partial image database was created for the third floor of the Computer Science Department building, to include sixteen rooms, staircase, a bridge and the corridors. Creating the database is a slightly tedious process. It takes around an hour to construct the database for a room, which includes the time for database construction, pruning as well as annotation of images. The database was constructed over multiple days. Database construction as well as experiments were carried out during the day on weekends. In all, the database contains around 300 locations (i.e corners). The goal of the experiments was to test the feasibility of the approach. We sought the answers to the following two questions : (1) How successful is our approach in achieving room-level accuracy?, and (2) How successful is our approach in estimating the orientation and location of the user anywhere in the building?

Experimental Methodology.

Three main experiments were conducted. The experiments were carried out on two users with five trials per user. In the first experiment, the user would wear the phone

Figure 3.9: *Low-Resolution Pictures of a Few Rooms Taken from the Door*

Figure 3.10: *Low-Resolution Query Image Matches with Image 3*

as a pendant and enter the room as the camera took a picture. The image database would comprise of only pictures corresponding to user standing at the door of a room and facing inside, with ten images per door. This experiment was conducted to find the probability of success for room-level accuracy. Hierarchical and History-based approach does not apply to this scenario. Figure 3.9 shows the pictures of a few rooms that were experimented with. Figure 3.10 shows an example query image that matches best with Image 3 in Figure 3.9. In the second experiment, the image database comprised of only inside-room pictures of all rooms. The experiment was conducted with the user wearing the phone as a pendant and walking around inside rooms, sending a query image every four seconds.

Figure 3.11 shows inside-room pictures of a few rooms. Figure 3.12 shows example of a query image that matches best with Image 4 in Figure 3.11. Most of the rooms in the Computer Science building are around 4mx7m. The system reports user's location to quarter-room level accuracy (North-East, North-West, South-East, South-West), and one of the four orientations (facing North, facing South, facing East, facing West). The database consists of ten images per corner to account for varying user heights and angles, and sixteen corners per room (four corners per quarter corresponding to the four

Figure 3.11: *Low-Resolution Pictures of Different Corners*

Figure 3.12: *Query Image Matches with Image 4*

orientations). The third experiment was conducted with the complete database, as the user walked around on the 3rd floor. This includes corridors, staircase and the bridge in addition to rooms. We call this *corner-level* accuracy. During the experiments, the users would either stand stationary when the image was being clicked, or walk at very low speeds, in order to cover all the corners. This was necessary because of GPRS latency.

Experimental Results.

Table 3.1 shows the results for the three experiments. The results correspond to the ten trials. From the results, it can be concluded that for applications that require only room-level accuracy, the Naive approach would suffice. Corner-level accuracy corresponds to the most general case: the user walking inside rooms, across rooms and inside corridors. History-based approach clearly shows improved performance for this case. Hierarchical approach performs well most of the time, but completely fails sometimes and is, therefore, not reliable.

We found that the Color Histograms algorithm worked very well as compared to the other two, and was chosen as the primary method of image comparison in our system. We noticed that while most of the images resulted in accurate location, some of them

Table 3.1: *Success Probability for the Three Localization Experiments (Using Camera-phone)*

Approaches	Naive	Hierarchical	History-Based
Room-level accuracy	93%	N/A	N/A
Quarter-room-level accuracy	83%	96%	94%
Corner-level accuracy	50%	Non-deterministic	80%

Table 3.2: *Energy Consumption and Response Time for Sending an Image and Receiving Location Update*

Image Size	Avg. Response Time	Avg. Energy Consumption
5KB Image	720 msec	630mJ
128KB Image	4100 msec	3600mJ

were hard to recognize and were often confused with other similar images, bringing down the overall success rate. Even when a query image was incorrectly matched, the correct one was often among the top three matches, suggesting that if the approach were combined with another low-cost localization mechanism, extremely high accuracies could be achieved.

We worked with low-resolution JPEG images of size 5 KB. Experiments with high-resolution images (128 KB in size) resulted in approximately the same success rate as low-resolution images. By working with low-resolution images, we could save substantially on energy as well as response time. Table 3.2 shows the average response time (i.e latency) for experiments, as well as the average energy consumption on the phone for sending a query image and receiving one location update.

An experiment was carried out to study the effect of changes in the environment, namely: (1) movement of furniture, (2) presence of human beings and (3) varying lighting conditions. This experiment was conducted for one room. We found that slight movement of furniture, such as chairs, computers, objects on tables and shelves did not affect the accuracy of results. The accuracy of results suffered when there was a significant change, such as removal of a desktop monitor from a desk. The accuracy fell drastically when lighting conditions were varied. When the lights in the room were dimmed, the success probability fell to less than 20%. Much to our surprise, the

Figure 3.13: *Left: Image in the Database; Center: Image With a Person; Right: Person Wearing a Brown Jacket*

presence of human beings in the pictures did not affect the accuracy significantly. Out of the sixteen corners in the room, only six suffered from the presence of a human being. Three of these were recognized with a success rate of only 10%, two were recognized with a success rate of 50%, and one was recognized with 90% success rate. For the last one, the success rate fell to 70% when the person in the picture wore a brown jacket. (Figure 3.13 shows the three pictures: the original picture, picture with a person in it, picture with a person wearing a brown jacket.)

It is not completely clear why some images are recognized more accurately than others in the presence of a human being, but we conjecture that it is because of the fact that some images have more distinguishing objects in them as compared to others. When an image is recognized with low accuracy due to presence of a person, it is almost always confused with an image of the same corner but taken from a distance. Some of these observations can be attributed to our primary method of image comparison, which uses Color Histograms. Color histogram of an image is produced by discretising the colors in the image into a number of bins, and counting the number of image pixels in each bin. Color histogram of an image varies only slightly with the angle of view and the movement of objects inside the image.

The computation time on the location server is 100-120 msec which is a fraction of the total response time. Response time is dominated by the time taken to send the image to the server and is therefore subject to the bandwidth delivered by the cellular data service, which in our case is GPRS. GPRS delivers a bandwidth of 20-40Kbps, depending on user speed. With the advances in the telecommunications technology,

significantly better bandwidths are expected. For example, 3G delivers a bandwidth of around 400Kbps at pedestrian user speeds and over 2Mbps in fixed locations, which is much higher than the bandwidth provided by GPRS.

3.4.2 Experimental Results for Light Sensor Based Localization

Light illumination measurements were collected for two office buildings that belong to the Department of Computer Science. In all, data was collected for 27 rooms and the corridor (henceforth 28 rooms). Of these, 8 rooms have windows and 20 rooms have static lighting conditions (i.e no windows).

Experimental Methodology.

Two separate datasets were collected– one in which the light sensor was worn on top of a hat (as shown in Figure 3.4(left)) and the other in which the light sensor was worn as a pendant (as shown in Figure 3.4(right)). When the light sensor is worn as a pendant, lower accuracy can be expected because the user's body obstructs and affects the amount of light energy incident on the light sensor. For either experiment, it only takes a minute to collect data for a room, and a total of half-an-hour to collect data for all the rooms. For rooms without windows, multiple such datasets were constructed over a time-period of one week. These datasets were mixed; some were used for training, and the others were used for testing. For rooms with windows, data was collected in a similar fashion but at three different times of the day (for seven days) and over a time period of three weeks, and mixed the data by hour of the day. The reason for collecting data at different times of the day and over several days was to observe the effect of diurnal cycle and weather changes on the accuracy of the approach.

For collecting testing data, the user walks through the rooms and corridors at normal walking speed, stopping at certain locations periodically to imitate real-life mobility. The data is then pre-processed using two data filters. The first filter fragments the dataset into smaller pieces, where each piece contains the data of a room. For this, the filter looks for jumps in the light intensity readings, which correspond to the user exiting/entering a room. The second filter discards contiguous repeated readings, which correspond to the user standing at a location.

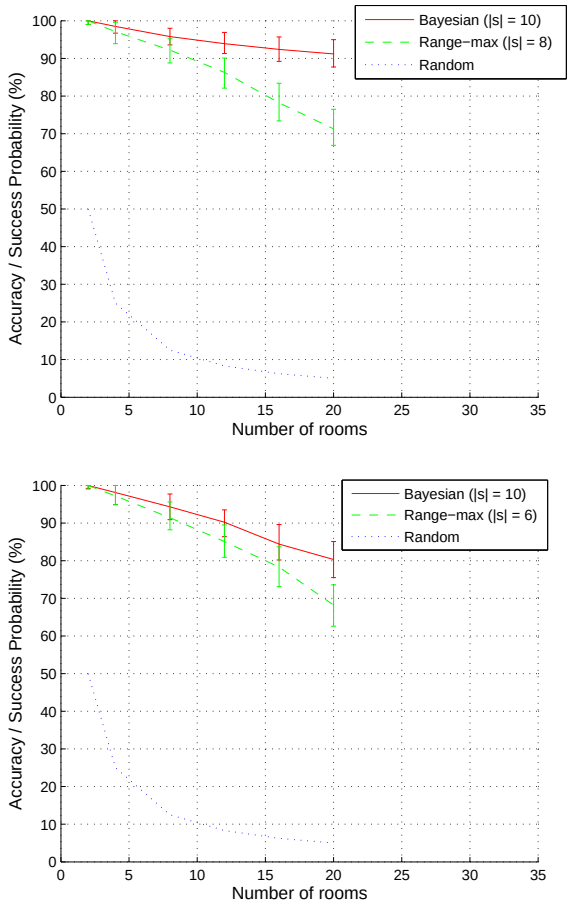

Figure 3.14: *Rooms Without Windows. Light Sensor Worn atop a Hat (top) and as Pendant (bottom)*

The goal of the experiments was to validate the following hypothesis: *it is possible to probabilistically determine which room the user is in if they are wearing a light sensor*. For validating this hypothesis, two fingerprinting/localization techniques, namely Bayesian and Range-max, were evaluated on two different datasets: one in which the light sensor is worn on top of a hat, and the other in which the light sensor is worn as a pendant. The scalability of these techniques is also tested by varying the number of rooms from 2 to 20 (for rooms with no windows), and from 2 to 8 (for rooms with windows). The ability to tell between 20 rooms or less can be sufficient for several applications if we assume that indoor localization solutions would typically work in a hierarchical fashion, i.e identify the building, identify the floor, identify the room.

Training and testing datasets were collected and stored separately. Training data was processed to obtain fingerprints for rooms using Bayesian and Range-max fingerprinting algorithms. Testing data was pre-processed to separate out light measurements for different rooms, and eliminate contiguous repeated readings. The testing data corresponding to a particular room was then fed into the two localization algorithms. Both the algorithms pick a contiguous set of points s starting at a random position in the dataset, which is then matched against the fingerprints as described in Section 3.3. This is repeated for all the rooms 100 times.

Localization accuracy would depend on $|s|$, which represents number of light illumination readings needed to localize the user. If the time interval between two consecutive light illumination readings is t, then the localization latency is given by $t|s|$. In our case, $t = 0.5$ second. Higher the value of $|s|$, higher the localization latency.

Results for rooms without windows.

Figure 3.14 shows the localization accuracy of the two algorithms as a function of the number of rooms. The graph on the top shows the performance of the algorithms when the light sensor is worn on top of a hat, with $|s| = 10$ and $|s| = 8$ for Bayesian and Range-max respectively. The average accuracy of Bayesian for 20 rooms is 91.2% and that of Range-max is 71.6%. The bottom graph shows the localization accuracy of the two algorithms when the light sensor is worn as a pendant, with $|s| = 10$ and $|s| = 6$ for Bayesian and Range-max respectively. The average accuracy of Bayesian for 20 rooms is 80.3% and that of Range-max is 68.1%. Any sensible localization algorithm should do better than *Random*, which determines user's location by randomly picking a room.

Figure 3.15 shows the localization accuracy of the two algorithms for 20 rooms, as a function of $|s|$. The graph on the top corresponds to the case when the light sensor is worn on top of a hat. Bayesian attains peak accuracy when $|s| = 10$, while Range-max attains peak accuracy when $|s| = 8$. The bottom graph corresponds to the case when the light sensor is worn as a pendant. Bayesian attains peak accuracy when $|s| = 10$ in this case, while Range-max attains peak accuracy when $|s| = 6$. As mentioned before, lower value of $|s|$ implies lower localization latency.

The localization accuracy is lower when the light sensor is worn as a pendant because

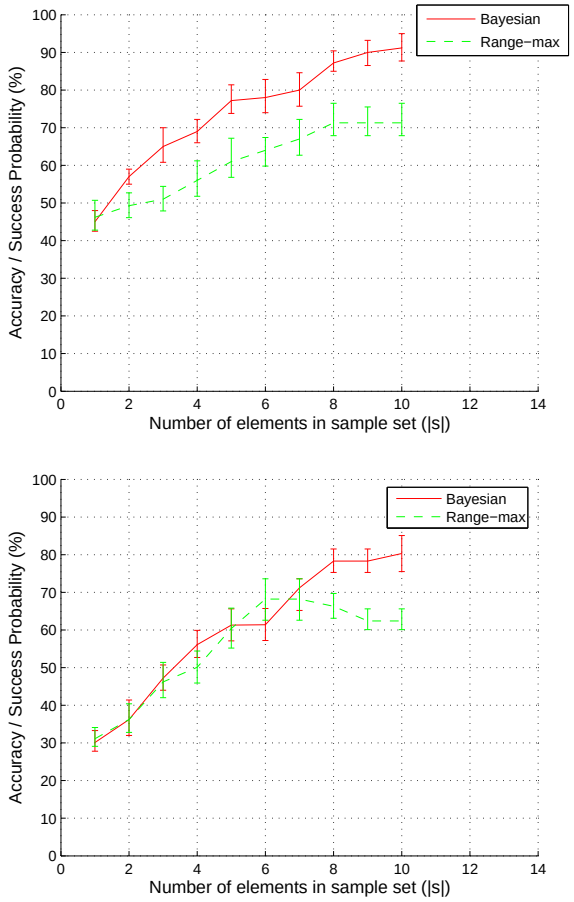

Figure 3.15: *Rooms Without Windows. Light Sensor Worn atop a Hat (top) and as Pendant (bottom).*

the user's body affects the amount of light energy incident on the light sensor. Bayesian consistently outperforms Range-max. However, Range-max requires a lower value of $|s|$ (hence lower localization latency) to reach its peak localization accuracy. Also, Range-max is independent of the amount of time the user spends at a particular location. For Bayesian, filtering needs to be carried out.

Results for rooms with windows.

The experiments were carried out in 8 symmetric rooms with large windows. Data was collected at three fixed times of a day (morning, noon and afternoon) for seven days over a time period of three weeks. The datasets were mixed and tested in several different ways to understand the effect of sunlight, diurnal cycle and weather changes

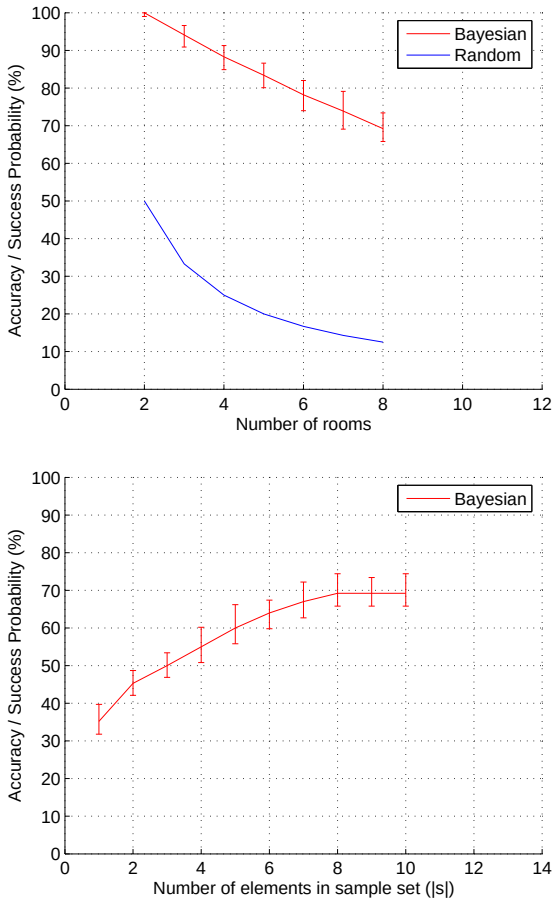

Figure 3.16: *Rooms With Windows. Light Sensor Worn atop a Hat*

on the accuracy of the approach. In this dissertation, we report the results obtained with Bayesian while wearing the light sensor on a hat. In one experiment, the data for a particular hour of a day was tested against the data for a different hour of the day for the same day. The accuracy was found to be between 35-42% for 8 rooms. In another experiment, the training data of all the days was mixed together and was tested against a separate dataset corresponding to one particular day. The accuracy for 8 rooms was found to be between 40-45%. In a third experiment, data for a particular hour for a certain day was tested against the data for the same hour for another day. The accuracy was found to be between 55-60%. Finally, the data for a particular hour for different days was mixed together and tested against a separate dataset for the same hour of a different day. This yielded the best accuracy, ranging between 68-73% for 8

rooms. Figure 3.16 shows the detailed results corresponding to this case.

From these results, we can conclude that best accuracy is achieved by including time information in the query and conditioning on the hour of the day. We can also conclude that the effect of weather can be partially neutralized by mixing the training data of several days together (conditioned on the hour). However, different sets of fingerprints would have to be constructed for different seasons. We noticed that even when a room is misclassified, it is almost always the second or third best candidate suggesting that the accuracy can be further improved if this approach is combined with another approach.

3.5 Discussion

The two localization solutions described in this chapter have their own limitations. The camera-phone-based localization scheme is not resilient to changes in the environment such as movement of furniture, and fails when the line of slight is blocked. Light-sensor based localization works moderately well in the presence of sunlight. As such, this approach is applicable at night or to buildings with significant number of window-less rooms, such as museums or girthy office buildings where only peripheral rooms have direct exposure to sunlight.

We believe that a practical solution for localization would be a fusion system that incorporates an array of sensors (such as camera, light sensor, GPS, WiFi interface, accelerometer etc), derives location information from each of the sensors and fuses them together to obtain accurate results. Although each of the sensors could fail under certain conditions (for example the light sensor may not work very well when there is interference from sunlight), the fusion system as a whole should be quite robust.

3.6 Summary

In this chapter, we presented two solutions for indoor localization: camera-phone based and light-sensor based. These solutions do not require any extra infrastructure to be installed in the environment.

The camera-phone based system was able to attain room-level accuracy with more

than 90% success probability and corner-level accuracy with more than 80% success probability, using a history-based location determination approach. The latency of receiving location updates over a GPRS connection is around a second, which would be even lower for a 3G connection. The image comparison algorithms remain unaffected if resolution of the images in the database is the same as the resolution of the query image. By using low resolution images, it was possible to reduce energy consumption and response time significantly.

Light-sensor based localization achieves up to 90% success probability under static lighting conditions, but works moderately well in the presence of sunlight. Construction of Bayesian fingerprints is fast and convenient. It takes around a minute to fingerprint a room.

Both the localization solutions presented in this chapter are infrastructureless and can be combined together to attain higher accuracies. For example, the knowledge of which room the user is in can significantly prune down the search space for camera-phone-based localization scheme. The light sensor could either be embedded in the phone or iPod of the user or worn separately as a clip.

Camera-phone based localization is a power hungry system that can drain the phone battery. With multiple such processes running on the phone, it becomes necessary to manage the limited battery lifetime. In the next chapter, we describe how the limited battery lifetime of mobile devices can be managed across different applications.

Chapter 4

Location-aware Battery Management

Location-aware personal computing is centered around the idea of running client frontends and applications on resource-constrained personal devices (e.g smart phone), many of which would run as background tasks including daemons for determining user's location, daemons for listening to incoming network requests, daemons for warming the phone cache etc. These processes would compete for the limited resources on these devices. The most crucial resource, which determines the availability of a personal device, is battery lifetime. We study the case of smart phone, which in this dissertation, represents the intelligent personal device. We show that location information can be used to manage battery lifetime intelligently.

On smart phones, the interfaces that inform the user of the battery levels (such as ACPI [1]) have not kept up with the evolution of the capabilities of these devices. A simple "battery remaining" or even "time remaining" does not enable the user to make the right decisions as to the spend of the energy budget and the request for recharging.

Prior research on dealing with the limited battery lifetime problem has focused on optimizing energy at different levels of the stack, starting with hardware all the way up to the application layer. The most commonly used energy optimization mechanism is *hibernation*, in which the CPU or other hardware units of a system are put in low-power mode when not being actively used. This is commonly done by the operating system. Dynamic voltage scheduling is another commonly used optimization technique in which a compiler-directed algorithm identifies program regions where the CPU can be slowed down with negligible performance loss [46, 40]. Two effective application-level energy optimizations have been proposed: cyberforaging [23] and application adaptation [32, 20]. In cyberforaging, computation is off-loaded from the battery powered mobile device

to a wall-powered server whenever profitable. In application adaptation, the fidelity of an application is lowered whenever low battery levels are detected. While a lot of work has been done on optimizing energy, there has been limited research on managing battery lifetime across applications and treating energy as a first-class operating system resource. To the best of our knowledge, *ECOSystem* [86] is the only piece of work that takes this approach. ECOSystem is a framework for sharing battery lifetime as a resource across competing applications.

The primary purpose of the mobile phone is telephony (i.e making and receiving calls and exchanging text messages). Other applications are secondary. Hence, there is a notion of *priority* among applications, which should be imposed on the manner in which applications consume limited battery power. A low priority application (such as a background task) should not be allowed to compromise the availability of a high priority application (such as telephony).

The key observation is that the knowledge of when the user will recharge the phone next is crucial to managing battery lifetime on phones. This is because the knowledge of when the next recharge will happen determines the total battery lifetime available to applications. A user study shows that users typically charge their phones at fixed locations [25]. Hence, by predicting the whereabouts of the user, it should be possible to predict charging opportunities. In this chapter, we describe a location-aware battery management architecture [66, 63] for smart phones (henceforth CABMAN), based on four principles:

- The availability of crucial applications to users should not be compromised by non-crucial applications.
- The opportunities for charging should be predicted to allow devices to determine if they have scarce or plentiful energy, instead of using absolute battery level as the guide.
- Location information can be used to predict charging opportunities.
- User involvement should be minimal.

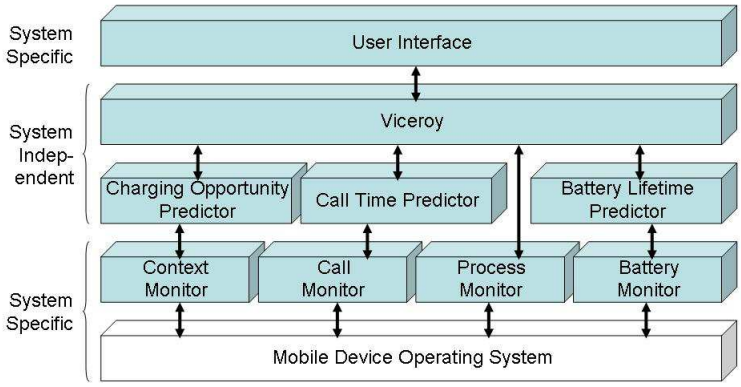

Figure 4.1: *Battery Management System Architecture*

4.1 Architecture and Algorithms

The key role of the battery manager is to be able to answer the query: *Will the phone battery last until the next charging opportunity is encountered?* In order to answer this query without involving the user, the battery manager should be able to answer the following three questions (1) when the next opportunity for recharging will be available and hence, what is the total battery lifetime available to the user? (2) what fraction of this battery lifetime will be consumed by critical applications such as telephony? and (3) what fraction of this battery lifetime can be left for use by non-critical applications?

We identify three subproblems: (1) prediction of the next charging-opportunity, (2) prediction of the call-time that might be required by the user in the interim, and (3) prediction of how long the battery will last if the current set of applications continue to execute on the device. With this knowledge, the battery manager will be able determine if the user will run out of battery sooner than they should, and ask them to terminate one or more applications or look for a charging opportunity.

In order to solve these three subproblems, we designed a system that can monitor

user context and sense battery level of the device; a set of algorithms for making predictions and a central component for assimilating the information together and warning the user appropriately. The system consists of eight components as shown in (Figure 4.1) divided into three categories: system-specific monitors, predictors, and the viceroy/UI.

System specific components: The *Process monitor* is responsible for keeping track of the processes running on the device and informing the viceroy whenever a new process is detected. *Battery monitor* probes the battery periodically and enquires about remaining charge and voltage level. *Context Monitor* is responsible for sensing and storing context information (such as location) on the phone. *Call Monitor* logs communication (incoming/outgoing calls, incoming/outgoing SMSs).

4.1.1 Charging Opportunity Predictor

The crux of the system is that a phone should determine if a charging opportunity is near enough for the battery to "last" until then (by which we mean maintaining the ability to execute the *crucial application*). If this is true, then the system should not inconvenience the user with unnecessary warnings or actions such as going into low-power modes with reduced functionality. If this is false, then even if the phone battery is relatively full, the system should warn the user that they risk a dead battery.

The charging opportunity predictor predicts the future whereabouts of the user from the location trace stored on the phone. Certain locations are marked as charging opportunities (based on the charge trace stored on the phone). The prediction of when marked locations will be reached is accomplished by pattern-matching the current pattern of location movements against the set of location traces stored on the phone.

Intuitively, the algorithm proceeds as follows. If the current location samples are, say, ABC, then a search is made through the history of all traces that contain the sequence ABC (say $DEABCFG$), and for each of the resulting traces, the time is determined between the time of entry of the current location and the time of entry of the next charging-capable location. These times are then averaged to get an estimate of *time-until-charging-opportunity*.

Formally, let a location trace tr be denoted as a sequence of tuples: *(l,t,c)*, where l

denotes location, t denotes the time when the user entered that location and c is either 1 or 0 depending on whether or not the user can charge the phone at that location. Let the j^{th} location trace tr_j be denoted by (l^j, t^j, c^j), let the i^{th} tuple in this trace be denoted by (l_i^j, t_i^j, c_i^j), and let the last tuple in this trace be denoted by $(l_{last}^j, t_{last}^j, c_{last}^j)$. For a given location trace (l^j, t^j, c^j), let t_{charge}^j denote the time when the user enters the next charging-location. A location trace tr_j is said to be *contained* in another location trace tr_k if $\exists i(l_0^j = l_i^k, l_1^j = l_{i+1}^k, .., l_{last}^j = l_{i+last}^k)$. Given a location trace $(l^{today}, t^{today}, c^{today})$, the time interval before the next charging opportunity becomes available is estimated as $(\sum_j (t_{charge}^j - t_k^j))/N | (contained(today, j), l_k^j = l_{last}^{today})$, where $contained(today, j)$ implies that tr_{today} is *contained* in tr_j, and N is the total number of traces for which $contained(today, j)$ is true.

4.1.2 Call Time Predictor

Telephony is regarded as the "crucial application" for mobile phones in that users always want to be able to use this application (e.g. for emergency calls or rendezvous with friends). The "non-crucial applications" should not be allowed to drain the battery to the stage where the user is deprived of telephony service.

To protect the availability of telephony, we need to predict the call time needs of the user. There are three options in order to achieve this. A simple method would be to ask the user could set a minimum call time level which they always like to maintain. However, this is not dynamic, so a user must continually ensure this setting is up-to-date. A more complex but dynamic method, which does not require user input, is to use past calling behaviour to find the average number of minutes of call time that the user needs during each hour of the day, and to use this to get an upper bound on the total call time required within a given time interval. This can be enhanced by viewing weekdays and weekend days separately, since call behaviour is likely to differ in that time.

Formally, let a call-trace *calltr* be denoted as a sequence of tuples: *(h,t)* where h denotes the hour of the day and t denotes the call-time used in that hour. Let the j^{th} call-trace $calltr_j$ be denoted by (h^j, t^j), let the i^{th} tuple in this trace be denoted

by (h_i^j, t_i^j). The call-time to be used in a given hour h_k is estimated as $(\sum_j t_k^j)/N$, where $calltr_j$ is a past call-trace, and N is the total number of past call-traces selected (e.g over the last three months). This algorithm is tested in the evaluation section and shown to work well.

The third option is to use a hybrid of the two listed above to achieve a conservative prediction. For instance, we could use the policy "keep twice my average call time available, and a minimum of 10 minutes for emergencies in addition to the predicted call time". The correct choice of policy depends on user preference, such as the user's perceived annoyance at being unable to make a call versus their perceived annoyance at having to charge their phone more often (which is the tradeoff being contemplated).

4.1.3 Battery Lifetime Predictor

Battery management software and hardware have developed recently and may provide accurate estimates of the charge level left in the battery. However, this by itself is insufficient to predict lifetime accurately, since applications may vary their battery demands over time. Also, batteries have different chemistries with different reactions to types of load (constant, spiky, etc) and they age. In this chapter, we propose a battery lifetime metric that is independent of battery age and takes into account applications' battery usage.

One obvious and often used method of predicting battery lifetime is to monitor the rate of drain of the battery and extrapolate this linearly to exhaustion. However, due to the factors mentioned above, this is not reliable. To illustrate this, we examine the battery discharge curves of a number of devices in idle mode (i.e. not running any applications, but with no power-down power management software active). We call this diagram the *base curve*. Figures 4.2 and 4.3 show base curves for a new laptop, an old laptop and an HP iPAQ. Since people do not replace old batteries, either by choice or due to affordability, we found it important to include an old battery in the experimental setup.

As we can see, while the base curve of new laptop looks linear, that of an old laptop is highly non-linear (consistently so over many iterations). At the same time,

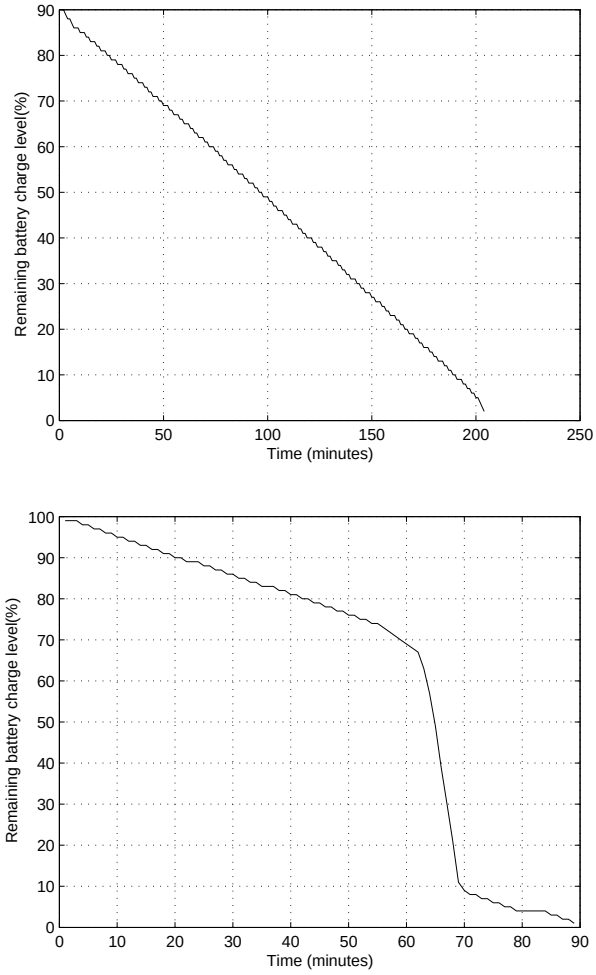

Figure 4.2: *Base Curve for a New HP Laptop (top) and Old Dell Laptop (bottom)*

the base-curve of the PDA (which reported just the voltage level) was also non-linear and spiky.

The approach we took is to compare the actual discharge when applications are running against the measured base curves in order to predict for battery life. This requires a one-time offline measurement of the base curve, which a device can do for itself during a period where the user does not need it (similar to battery reconditioning that users perform today). This can be repeated periodically (e.g. on the order of months) to make sure that the changing performance of the battery over its lifetime is compensated for. The algorithm proceeds as follows: with a given set of applications

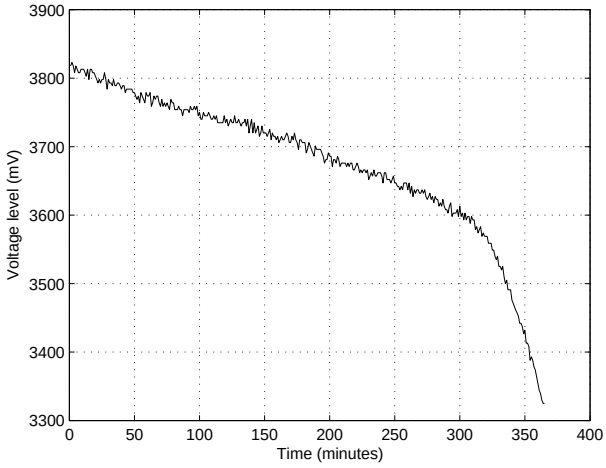

Figure 4.3: *Base Curve for an HP iPAQ*

running, we measure the *discharge speedup factor* over a particular drop in battery energy (or voltage). This is calculated by comparing the time it would take for the battery to reduce by that amount during idle (from the base curve), divided by the actual time that it took for the battery to drop by that amount. In other words, with applications running, we measure the battery capacity c_1 and c_2 at two time instances t_1 and t_2 respectively; we find time instances t_3 and t_4 that correspond to battery capacities c_1 and c_2 on the base curve. The *discharge speedup factor* is calculated as $(t_4 - t_3)/(t_2 - t_1)$.

We then divide the remaining lifetime of the battery if the device were idle (from the base curve) by the discharge speedup factor, to obtain the predicted remaining time for the battery. As we shall see in the evaluation section, this has proven to be a very accurate measure of battery lifetime remaining.

4.1.4 Viceroy and User Interface

The viceroy is CABMAN's central component. It uses input from the predictors and directly from the process monitor, in order to decide when action must be taken using some form of user interface (UI). The main job of the viceroy is to continually monitor whether the battery lifetime prediction combined with the battery requirement of the

estimated call time requirement from the call time predictor, means that the battery will expire before or after the next charging opportunity. If the energy level is not sufficient to last until the next charging opportunity, then the viceroy must use the UI (audible or visual signals) to notify the user. Formally speaking, the user should be warned if $t > r - f(m)$, where t is an estimate of the time interval before the next charging opportunity surfaces, r is an estimate of the remaining battery lifetime, m is an estimate of the required call-time and $f(m)$ is the map from call time to battery lifetime.

When warned, the user may be able to take care of the situation by (i) killing some battery-hungry applications (up to and including powering down the device as a whole), (ii) change their behaviour so as to make less power demands of the device, (iii) plan to charge the device according to the time-scale that the viceroy predicts the device will last, or (iv) accept and understand that they may lose the ability to make calls (or, in general, execute critical applications).

When the user is *at* a place with a charging opportunity (as may often be the case, e.g. for users with a home charge and office charger), the viceroy's job is to decide whether to use the UI to ask the user to charge the device, or whether the battery level is sufficient to reach the next charging opportunity.

4.2 Experimental Results for the Battery Management Architecture

We have implemented a battery management prototype for Linux and Symbian OS. The purpose of this prototype was to carry out a feasibility study. The ability to make predictions with acceptable errors is a key indicator of the real-life performance of the battery manager. Therefore, for evaluating the system, we evaluate the three prediction algorithms used: charging opportunity predictor, call-time predictor and battery-lifetime predictor.

Charging Opportunity Predictor

Charging opportunity predictor predicts when the next opportunity for charging the phone will be encountered. To evaluate the performance of the charging opportunity

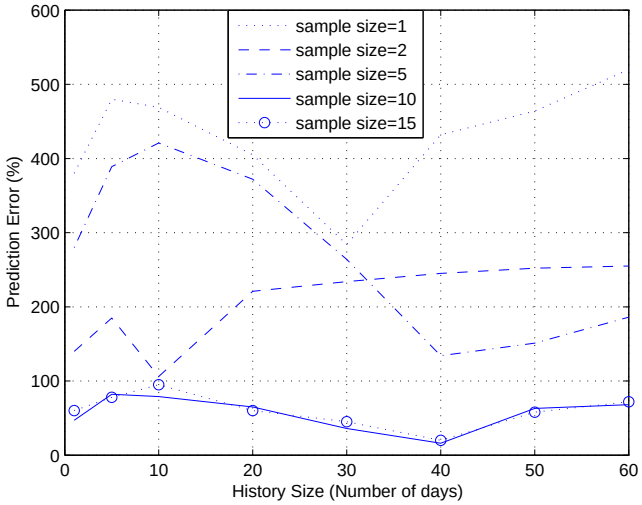

Figure 4.4: *Charging Opportunity Prediction Error for Various Sample Sizes and History Sizes*

predictor and call time predictor, we used a large trace of measurements of real users captured in MIT's Reality Mining project [3], which in turn used context logging functionality from the University of Helsinki. This excellent data set was gathered by deploying Nokia 6600 phones to more than 80 subjects for around nine months.

We present the percentage prediction error averaged over the traces of all subjects, varying our algorithm's parameters of *sample size* and *history size*, in Figure 4.4. As we expect, an increasing sample size generally increases accuracy (the curve is lower) and reliability (the curve wavers less), as more closely-fitting historical data is used in prediction, with a sample size of 1 being only the current location used, and a sample size of 15 being the previous 15 locations. The effect of increasing sample size appears to bottom out at around 10. From a tested range of 1–60 days, history size appears to be optimal at 40 days; intuitively more history provides a longer averaging period, but longer-term changes in user behaviour (e.g. the change of home or workplace) mean that use of too much history has a negative effect.

Using the parameters of 10 samples and a 40 day history, the average prediction error across the 80 user, 9 month trace is 16%, which corresponded to an absolute error of 12 minutes on average. This indicates that the charging opportunity prediction

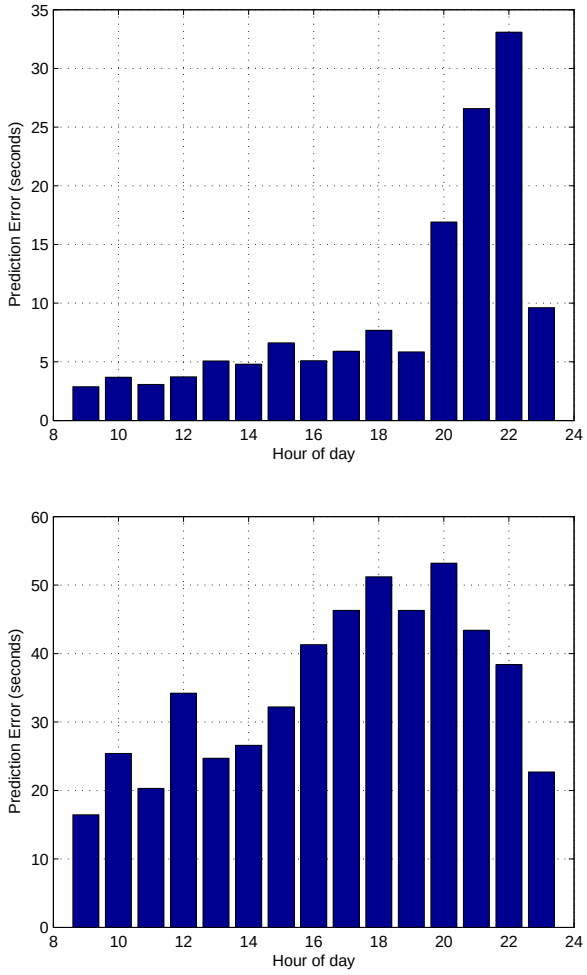

Figure 4.5: *Absolute Call Time Prediction Error for Weekdays (top) and Weekends (bottom)*

algorithm is highly likely to give useful results in practice.

We also evaluated the algorithm by conditioning it on the day type (weekend or weekday), that is, if the current location trace is that of a weekday then it is compared against the past location traces of only weekdays. We did not notice any significant difference in performance.

Call Time Predictor

Call time predictor predicts how much call-time will be required by the user in a given window of time. The Reality Mining traces are also used to evaluate the call time

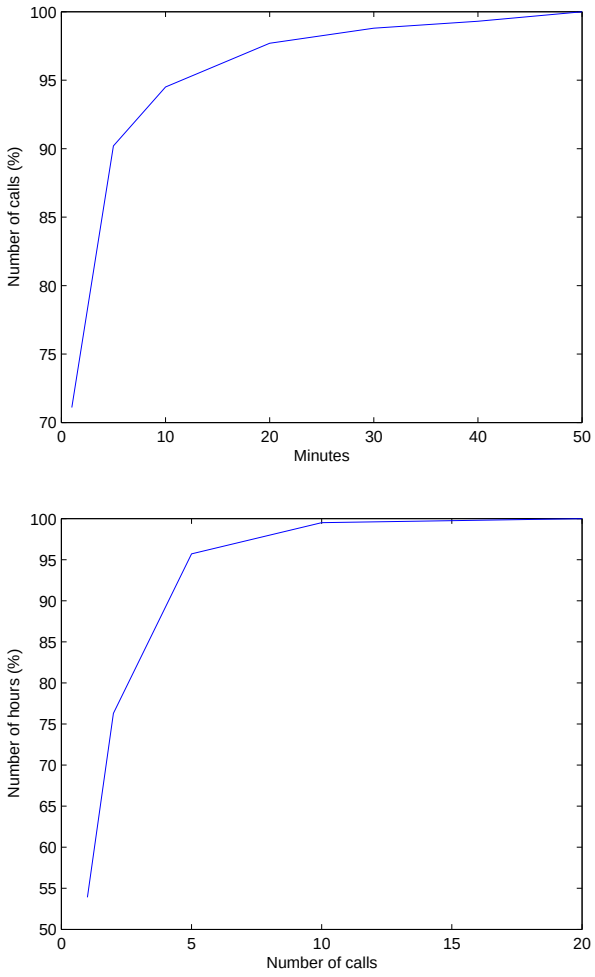

Figure 4.6: *Cumulative distribution function of the Length of Phone Calls (top) and the Number of Calls Made During Each Hour (bottom)*

predictor. Using the call logs of these traces, we can execute the prediction algorithm described previously, where the number of minutes of call time used in a given hour of the day is predicted by the average number of minutes used in previous days. The average prediction error of this algorithm is shown in Figure 4.5. We can see that, as one might expect, it is easier to predict call time requirements for the middle of the day than it is for evenings (where users typically make many calls), and easier for weekdays than weekends. However, even in the worst case the average prediction error is under a minute out of the hour.

The low prediction errors seem suspiciously good, until one examines the actual calling pattern of users, as shown in Figure 4.6. These cumulative distribution functions (CDFs) show that the typical call is short (71% calls are less than a minute, 90% calls are less than 5 minutes), and in a typical hour very few calls are made (75% with 2 calls or fewer), although both of these curves have a "long tail", which account for the occasional long call. While we obviously cannot account for the latter case (how can we predict an incoming call from an occasionally-in-contact friend when humans cannot?), the type of functionality we are looking to preserve in CABMAN is the crucial application of telephony for rendezvous or emergencies. The user does have a choice to specify the additional amount of call time they would like to reserve over and above the predicted call time.

Therefore, the short calls forming the majority of usage are those that interest us, and the algorithm presented is shown to provide a feasible way of dynamically estimating the call requirements of the users in this large trace. Particularly when combined with user-specified minimum thresholds for call-time-remaining (as discussed previously), we believe that the system can make useful estimates for user needs for the crucial application of telephony.

Battery-lifetime Predictor

In order to evaluate the performance of the battery-lifetime predictor, we experimented with three different machines: an HP laptop with a new battery, a Dell laptop with a very old battery and a regular HP iPAQ PDA. First, the *base curves* for all the three machines were obtained, as shown in Figures 4.2 and 4.3. Next, we obtained the actual discharge curves for a set of applications (web, music and video) and all combinations of these applications running together (7 combinations) on all three machines. We simultaneously ran the prediction algorithm for battery lifetime (using a 2 minute observation window to obtain a prediction), and monitored the device's own battery lifetime prediction via the ACPI interface. Advanced Configuration and Power Interface (ACPI) defines common interfaces for device configuration and power management on Linux-based systems.

As previously described, the prediction algorithm essentially calculates the *discharge*

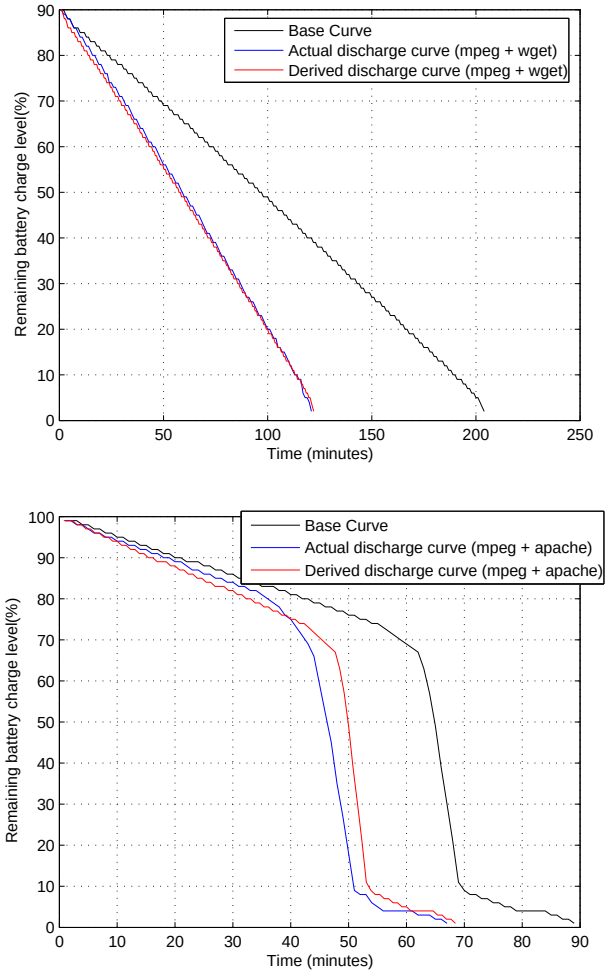

Figure 4.7: *Base Curve Together With Discharge Curves (actual and derived) for the New HP laptop (top) and Old Dell laptop (bottom)*

speedup factor based on observation of the running set of applications, and uses it to predict the remaining battery lifetime. In the process, the algorithm implicitly derives a predicted discharge curve for the given set of applications, which is useful in order to illustrate the performance of the algorithm.

Figures 4.7 and 4.8 show the base curve for the three devices together with the actual and derived discharge curves for a web application and movie player executing together. In all cases, the actual discharge curve and the derived discharge curve track each other closely, showing that the algorithm performs well. In the case of iPAQ,

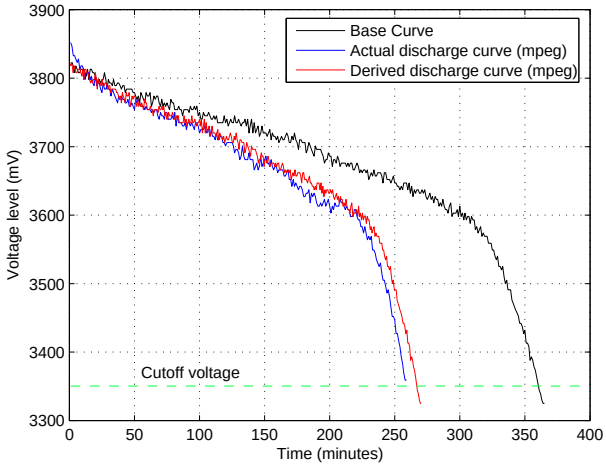

Figure 4.8: *Base Curve Together with Discharge Curves (actual and derived) for HP iPAQ*

Table 4.1: *Comparing Accuracy of Our Battery Lifetime Prediction Algorithm with ACPI's*

Machine	Average Prediction Error	
	Our algorithm	ACPI
New HP laptop	1.2%	23.5%
Old Dell laptop	6.1%	120.2%
HP iPAQ	3%	52.2%

the discharge curve is a measure of the instantaneous voltage level as opposed to the battery charge level as in the case of laptops. The iPAQ fails when the voltage level falls below a certain threshold.

Table 4.1 summarizes the performance of the battery prediction algorithm against the devices' own battery lifetime predictors (obtained via ACPI), across experiments using all 7 combinations of applications. Our algorithm clearly wins, with average errors of 1%, 3% and 6%, while ACPI's estimates are (on average) over 100% off (i.e. predicting more than double the actual lifetime) in the case of an older battery with a non-linear discharge curve. In absolute terms, we are able to predict battery lifetime with an average accuracy between 4 minutes for the new laptop and 12 minutes for the iPAQ. This experiment shows that our battery prediction algorithm is capable of

providing accurate enough information to support the feasibility of CABMAN.

4.3 Discussion

Charging-opportunity predictor and call-time predictor perform reasonably well for an average user whose life entropy is not very high. For users with a very high entropy lifestyle, the prediction algorithms described in this chapter may not work very well. Additional context information will be needed to improve the accuracy of these prediction algorithms for such users. This may include calendar information, information about the travel plans of the user, context information of the contacts of the user, etc. Obtaining and using this additional context information can be challenging in itself.

Many users charge their phones in their cars while driving. For such users, it is not always possible to associate charging with location. The charging-opportunity predictor should then associate charging with the user's presence in the car, and should predict when the user will be in the car next. This would also require additional context information to be logged.

4.4 Summary

In this chapter, we described a system and a set of algorithms for managing limited battery lifetime on smart phones. The main idea is to log user's information on the phone (such as location, call-time usage, etc) and use it to predict future whereabouts of the user and in turn use this information to decide what fraction of the battery should be reserved for crucial applications. Experimental results demonstrate feasibility of this approach. The knowledge of user's future whereabouts can enable several non-conventional applications including mobile data pre-fetching and route planning in addition to battery management.

In the next chapter, I present a solution for preserving the location-privacy of the user against untrusted services.

Chapter 5

Information Flow Control for Location Privacy

Location-aware personal computing requires user location to be shared with untrusted third parties, such as web services. Releasing location information without breaching user privacy is a hard problem. Unless the users are secure about their location privacy, they would be reluctant to use location-aware applications. This hurdle would have to be overcome if location-aware computing is to be globally accepted. This problem has been studied in the past and solutions proposed [31, 43, 36]. The state-of-the-art solution applies the idea of k-anonymity [77, 36] to location information using data perturbation techniques. In this solution, location information is perturbed by reducing the spatiotemporal resolution until the data met the k-anonymity [77, 36] constraint. This is a very generally applicable technique. However, it may lead to inferior quality of service if the accuracy of location information does not match the requirements of the service.

In this chapter, we describe a *service-specific* location privacy approach, centered on an information flow control analysis. Consider a scenario where a trusted location server maintains mobile subjects' position information. Location information consists of identity of the subject, its location, and the timestamp when the subject was present at that location. When an LBS provider needs location information, it can request to install an aggregation module on the trusted server. The module can access location records on the server and communicate aggregate results (e.g., distance between two cars, density of vehicles in region, or mean velocity of vehicles on road segment) to the LBS provider. This eliminates the need to reveal raw location information to the LBS as the desired result is computed on the trusted server itself using *exact* location information.

In order to preserve privacy, the trusted server needs to ensure that the data sent to the LBS is indeed aggregated and *location-safe*. That is, the LBS provider must be unable to deduce the location of any mobile subject from the received information. Since an aggregation module, due to malice or incompetence, may covertly leak information, this requires a detailed analysis of the aggregation module. This analysis can be performed either by the operator of the trusted location server or another trusted third party (like VeriSign) that guarantees correct behavior of the module. To enable the analysis and certification of a large number of different aggregation functions, there is a need for tools to assist in verifying code for location-safety. In this context, we describe a system based on an information flow analysis tailored to aggregation modules for time-series information such as location traces.

5.1 Information Flow Control

Information-flow policies are end-to-end security policies that provide more precise control of information propagation than access control models. A large body of work exists in this field, most of which is purely theoretical in nature. Here, we summarize the most noteworthy pieces of work in this area that have led to the evolution of this field. The lattice model for multilevel security systems was first proposed by Denning et al [29]. In this model, objects are assigned different security levels, where objects can be files, segments or program variables depending on the level of the detail. The security levels are organized as a lattice. Information is allowed to flow only from low security levels to high security levels. This has been the universally accepted model for representing multilevel security systems ever since. A special case is when there are only two security levels: *public* and *private*.

State of the art in information-flow control is *Non-interference* [26, 59], which was introduced by Goguen and Meseguer in their seminal paper [59]. Intuitively, non-interference requires that high-security information does not affect the low-security observable behavior of the system. In other words, private data does not influence

public data. Non-interference is fairly easy to enforce using language-based techniques [30, 51, 85, 52]. In particular, static program analysis has demonstrated advantages of little run-time overhead, the capabilities of managing implicit information flows, and provable guarantees. Jif [85, 52] is a popular static analyzer that enforces non-interference on annotated Java programs for multilevel security systems. Central to this implementation is the notion of a *principal*, which is an entity (e.g user, process, party) that can have confidentiality concern with respect to data. Principals can express ownership on data (i.e program variables) via *security labels*, which are analogous to security levels in multilevel security systems. These security labels are used to track flow of information between program variables. Illegal information flows are prohibited using type analysis.

Non-interference, however, is a very strict requirement for most systems where private data often interferes with public data. Its applicability is, therefore, extremely limited. This was explicitly stated in [71]. Since then, there has been research on relaxing non-interference. Giacobazzi et al proposed *Abstract Noninterference* [34], where they used abstract interpretations to generalize the notion of non-interference by making it parametric to what the attacker can analyze about the information flow. Many downgrading scenarios can be formally characterized in this framework. Downgrading/declassification is a term used for lowering the security class of an object, e.g from private to public. This allows sensitive information to be leaked when necessary. However, this framework is mainly theoretical. To practically apply this theory in building program analysis tools, we need to design ways to express security policies and mechanisms to enforce them. *Approximate Non-interference* [58] is a model in which the notion of non-interference is approximated in the sense that it allows for some exactly quantified leakage of information. This is characterized via a notion of process similarity. In this model, programs leaking more information are considered less secure. However, comparing the quantity of information leakage does not have direct sensible meanings in most situations.

In a recent work [49] Li and Zdancewic proposed a generalized framework of *downgrading policies*. A downgrading policy defines how and when the security level of a

particular sensitive data entity can be downgraded to allow leakage when necessary. Using this framework, the user can specify downgrading policies which are enforced using a type system. The main contribution of this work is the formalization of a framework that enables downgrading. The real challenge is to define downgrading policies that are applicable to real systems.

5.2 Non-Inference

The problem of verifying location-safety of an aggregation module can be regarded as an *information-flow control* problem. The standard information-flow control model called *Non-interference* [59, 26] requires that any possible variation in private data must not cause a variation in public data. That is, if the value of a public variable q depends on that of a private variable p then non-interference is violated. In effect, non-interference isolates private data from public data. In doing so, it guarantees that the publicly observable behavior of a system/program that does not reveal *anything* about its private behavior. However, data isolation is an extreme measure, and in many real applications it is not possible to isolate private data from public data. For example, in our case the migratory code computes results based on private location information. These results are transmitted back to the LBS and are public. Since private variables affect the value of public variables, it is not possible to enforce non-interference in this case. The question then is: Is it possible to envisage an information-flow control model that does not require data isolation and yet preserves privacy? Under what assumptions can such a model be realized?

In this chapter, we propose a weaker model of information-flow control called *non-inference*. *Non-inference* requires that the adversary should not be able to *infer* the value of a private variable based on the values of public variable.

Let us denote a program as a function f which takes inputs $i_1, i_2, ..i_n$ and produces outputs $o_1, o_2....o_k$. Let P denote the set of private input variables and Q the set of public input variables. All output variables are assumed to be public. Function f satisfies non-inference if and only if $\neg(\exists i_k \in P|\{f, o_1, o_2....o_k\} \rightarrow i_k)$. Inference is

denoted by the symbol $\rightarrow$. Informally speaking, a program satisfies non-inference if an adversary cannot infer the exact value of a private input variable from the output variables and the program code.

The question is: Is it possible to decide if a program satisfies non-inference? It can be shown that in the most general case (when no assumptions are made about the program or the capabilities of the adversary), non-inference is undecidable. (We provide a proof in the next section). However, non-inference is decidable if we can assume that multiple executions of untrusted code are independent of each other. (We prove this in the next section.) To understand this better, consider the following code snippet:

```
int h(int x1, int x2, int i){
    int out=0;
    if(i == 1){
       out = (x1+x2)/2;
       output(out);
    }else{
       out = x1*x2;
       output(out);
}
```

Here *x1* and *x2* are private variables and i is a public variable. Function h returns the average of *x1* and *x2* if $i = 1$, otherwise it returns the product of the two variables. Body of function h and the value of the input variable i are sent by the untrusted service. From one execution of h, the untrusted service learns either the product or the average. It cannot infer the value of *x1* or it x2 from *out*. Now, consider two different executions of h with i=1 and i=2, respectively. The first execution returns average of *x1* and *x2*, while the second execution returns product of *x1* and *x2*. By knowing both average and product, the adversary can infer the values of both *x1* and *x2*.

Non-inference, therefore, allows information to flow from private variables to public variables, but in order to prohibit the adversary from learning the value of any private variable from public variables, it is necessary to assume that multiple executions

are independent. Execution independence requires that the results obtained from one execution should not aid the results obtained from another execution of the same or different application. This is usually the case for location-based applications. In this example, if *x1* and *x2* are x-coordinates of two moving vehicles, and there is a time gap between the two executions of h, then the values taken by *x1* and *x2* would be different for the two executions. This makes the two executions of h independent of each other. Non-inference is therefore applicable to location-based mobile applications.

5.3 Proofs for the Theoretical Properties of Non-Inference

Model of Protection Systems: For sake of completeness, we first introduce the protection system as described in [38], which consists of the following parts:

- a finite set of generic rights R
- a finite set C of commands of the form:

```
command c(X1, X2, X3,...Xk){
   if r1 in (Xs1, Xo1) and
      r2 in (Xs2, Xo2) and
      ...
      rm in (Xsm, Xom)
   then
      op1
      op2
      ..
      opn
end
```

where op_i is one of the primitive operations

enter r into (X_s, X_o)

delete r from (X_s, X_o)

create subject X_s

create object X_o

destroy subject X_s

destroy object X_o

Also, $r_1, r_2, ...r_m$ are generic rights $s_1, s_2, ...s_m$ and $o_1, o_2, ...o_m$ are integers.

A configuration of a protection system is a triple (S, O, P), where S is the set of subjects, O is the set of objects, $S \in O$, and P is an access matrix, with a row for every subject in S and a column for every object in O. $P[s, o]$ is the set of rights that subject s has over object o. An operation such as *enter r into* X_s, X_o*)* will enter right r in the access matrix at position (x_s, x_o), where x_s and x_o are actual parameters corresponding to formals X_s and X_o. An example of subjects and objects would be processes and files respectively.

Transition from a configuration $Q = (S, O, P)$ to another configuration $Q' = (S', O', P')$ under the execution of an operation *op*, is represented as $Q \vdash_{op} Q'$. Reachability of Q' from Q is represented as $Q \vdash_* Q'$.

Safety: Given a protection system, a command $c(X_1, X_2, ...X_k)$ is said to *leak a generic right r from configuration* $Q = (S, O, P)$ if c, when run on Q can enter right r into a cell of the access matrix which previously did not contain r.

An initial configuration Q_0 is said to be *unsafe* for r (or *leaks* r) if there is a configuration Q and a command c such that

(1) $Q_0 \vdash_* Q$ and

(2) c leaks r from Q

Q_0 is safe for r if Q_0 is not unsafe for r.

In [38], Harrison et al prove that *it is undecidable whether a given configuration of a given protection system is safe for a generic right r.* In other words, *safety problem is undecidable.* In addition, they prove that *the safety problem without the create primitive is decidable and complete in polynomial space.* It is, however, not clear if a polynomial time solution is possible.

Complexity of Non-inference: We will show that the *safety problem*(as described in the previous section) can be reduced to the problem of deciding non-inference. Through this, we prove that non-inference is undecidable. In addition, we reduce the problem of deciding non-inference for single execution of a program to the *safety problem without create primitive*. Through this, we prove that non-inference for uni-directional information flow is decidable.

Theorem: *Non-inference for an arbitrary program is undecidable.*

Proof: We reduce the safety problem to the problem of deciding non-inference. Given an initial configuration $Q = (S, O, P)$, we create a program M with a set of variables V such that $V = S \cup O - \{S_1, S_2\}$. Input variables $V_{input} = \{v|(S_1, v) = r\}$, where (x, y) represents the cell in matrix P corresponding to row of x and the column of y. Output variables $V_{output} = \{v|(S_2, v) = r\}$. Intuitively, S_1 can be looked upon as a subject that has rights over all input variables and S_2 as a subject that has rights over all output variables. All input variables in set S are labeled private, while those in O are labeled public. The statements of M are created by converting each row of the access matrix into an assignment statement. The assignment statement corresponding to the row for subject S_i is $S_i = \{\oplus\{S_j|P_{ij} = r\}\} \oplus \{\oplus\{O_k|P_{ik} = r\}\}$, where $\oplus$ corresponds to the concatenation operator.

Program M when executed, will create a dependency matrix for its variables which will correspond to the access matrix P. To understand this, note that an assignment statement in M of the form $v_1 = v_2$ essentially corresponds to a right r in the cell (v_1, v_2) of the matrix P. Thus, rights in P represent data dependencies in M. If M violates non-inference, its execution will lead to a configuration such that by executing some command c, the adversary can enter a right r into a cell (S_2, x), where x is a private input variable of M. If M satisfies non-inference, then no such command can be executed. The problem of deciding safety for right r in configuration Q can be thus solved by creating a program M from Q and deciding non-inference for M.

Theorem: *Non-interference for uni-directional information flow is decidable.*

Proof: We reduce the problem of deciding non-inference for single execution of a program to the safety problem without the create primitive. Safety problem without the create primitive has been shown to be decidable [38].

Given a program M, let V be the set of variables of M and E the set of expressions in M. We create a configuration $Q = (S, O, P)$, such that $S = O = V \cup E \cup In \cup Out$ where In is the placeholder for a subject that has rights over all input variables of M and Out is the placeholder for a subject that has rights over all output variables of M.

Program M (in the intermediate representation) can have two types of statements: assignment statements or if-then-else statements. Matrix P is created from the statements of M. Assignment statements in M can be of three types: (1) $v_1 = v_2$ (2) $v = e$ (3) $e = f(v_1, v_2, ..v_n)$, where v_i represents a variable, e represents an expression and f represents a function that computes the value of an expression e from the variables occurring in it. P is created from the assignment statements in the following way: for every statement of the form $v_1 = v_2$, right r is entered in the cell corresponding to the row of v_1 and the column of v_2. Similarly, for every statement of the form $v = e$, right r is entered in the cell corresponding to the row of v and the column of e. For statement of the form $e = f(v_1, v_2, ..v_n)$ (e.g $e = x + y$), right r is entered in the cells corresponding to the row of e and the columns of $v_1, v_2, ..v_n$. All diagonal cells of the matrix have the right r in them. The row corresponding to In has right r in all the columns corresponding to the input variables of M. Similarly, the row corresponding to Out has right r in all the columns corresponding to the output variables of M.

Assignment statements are responsible for explicit information-flow in a program. Conditionals lead to implicit information-flow. For example: if($x = 0$) then $y = 1$ else $y = 2$, creates an information-flow from x to y, since the value of y depends on the value of x. To accommodate for implicit information-flow due to conditionals in M, P is extended in the following manner: for every control variable (e.g x in the above example), right r is entered in the cells corresponding to the column of the control

variable and the rows of all variables/expressions occurring on the left-hand-side of the assignment statements that are enclosed within the conditional corresponding to the control-variable. To understand this better, consider the following code snippet:

```
if(x = 0) then
   v1 = v2
else
   v3 = v4
end
```

Here, x is the control variable. $v1$ and $v3$ occur on the left-hand-side of the statements enclosed inside the conditional, and hence their value is affected by the value of x. For this example, right r would be entered in the cells $(v1, x)$ and $(v3, x)$ of P.

Configuration $Q = (S, O, P)$ thus created, captures the possible information-flow dependencies in program M. If some command c can be executed on the configuration Q such that right r gets entered in a cell (Out, v), where v is a private input variable, then this translates to an adversary learning the value of v. In that case, M would violate non-inference. If no such command can be executed, then M would satisfy non-inference. Non-inference for M can therefore be decided by deciding safety for the configuration Q.

The *create* primitive is used for creating new subjects or objects. What does the absence of *create* primitive correspond to in the non-inference domain? We informally argue that it corresponds to single execution of a program. For understanding this, it is necessary to understand what primitive operations and commands in the protection system domain correspond to in the non-inference domain. The command *enter r into (x, y)* in the protection system domain essentially corresponds to $x = f(y)$ in the non-inference domain, and represents information flow (or data dependency) from y to x. Similarly, *delete r from (x, y)* corresponds to $x = v$, where v is not a function of y. This results in destroying the data dependency between x and y.

The primitive *create subject s* or *create object o* essentially corresponds to creating temporary buffer variables in the non-inference domain, which an adversary can use to

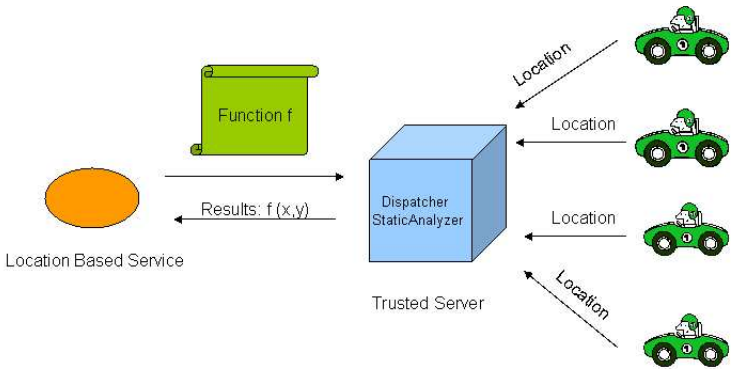

Figure 5.1: *Framework for Applying Non-Inference*

store values of output variables corresponding to an execution of a program. When the *create* primitive is taken out of the system, it implies that the adversary cannot create any temporary variables to hold values of variables resulting from one execution. This ensures that information that flows out of a program cannot be fed back into it. In other words, the program cannot be executed again with modified inputs that depend on the outputs of the previous execution. Multiple executions are possible, but they would be independent. This is what is implied by single execution or uni-directional information flow. Note that corresponding to commands that execute on configurations and result in violation of safety, are programs that an adversary would run on the output variables to deduce the value of a private input variable.

Thus, we can reduce the problem of deciding non-inference for single execution of a program to the safety problem without the create primitive, which is decidable.

5.4 Model

Figure 5.1 shows the bird-eye view of our model. In our model, the location information of mobile nodes is maintained on a trusted server. An LBS that needs to compute a result based on location information sends a piece of code to the trusted server along with some data, which is optional. The code executes on the trusted server, reads location information resident on the server, and the data sent by the LBS. It then computes some results and sends them back to the LBS. Without loss of generality, we

assume that the mobile code is a single function encapsulated in a Java class. A program with multiple functions can be reduced to a single function by inlining function bodies. We assume that all input and output variables are of type *byte* (8-bit integers) in order to make sure that exact values of two variables cannot be stored in one variable, which is important for the solution. The function is invoked from the *Dispatcher* class, which feeds the data sent by the LBS and the location information to the function. Before invoking the function, the *Dispatcher* class invokes the *StaticAnalyzer* that first checks to see if the function is pre-certified, if not it analyzes the function and determines if it satisfies non-inference. If the function satisfies non-inference, it is allowed to execute and read exact location information from the location database. If it fails to satisfy non-inference, low-resolution location information is fed into it, as suggested in [36]. The *Dispatcher* also ensures that there is a minimal time gap between two code executions in order to guarantee execution independence as described in the next subsection.

5.5 Deciding Non-Inference for Location Based Applications Using Static Analysis

The analyzer uses static program analysis to decide non-inference. The analysis consists of two phases. In the first phase, global data-flow analysis is used to construct information-flow relations between the variables(V) and expressions(E) of the code T under examination. From these information-flow relations, dependency information between private input variables ($P \in V$) and output variables ($O \in V$) is derived and stored in a matrix M.

In the second phase, the analyzer decides if the information-flow relations stored in the matrix satisfy non-inference. For this, the information-flow relations are treated as linear equations and the theory of solvability of linear equations is applied. This is the main idea used in deciding non-inference. We provide the details in what follows.

5.5.1 Information-Flow Relations

We define three information-flow relations for capturing data dependency in a program:
R_1 from V to E
R_2 from E to V
R_3 from, V to V.

$R_1(v, e)$ can be interpreted as "the value of variable v on entry to T *may be used* in the evaluation of expression e in T". For example, the code S:
if $(a > 10)$ then $(m = a + b;\ n = m * a)$ else $k = b$
contains three expressions: $\{a > 10, a + b, m * a\}$. The value of variable a on entry to this code may be used in evaluating all the three expressions, while the entry value of b may be used in evaluating only $(a + b)$. The entry value of m will not be used for evaluating any of the three expressions. For this code, we have: $R_1(a, a > 10)$, $R_1(a, a + b)$, $R_1(a, m * a)$ and $R_1(b, a + b)$.

$R_2(e, v)$ can be interpreted as "the value of expression e in T *may be used* in obtaining the exit value of variable v". In the previous example, both expressions $(a > 10)$ and $(a+b)$ may be used in obtaining the exit value of m. Similarly, all the three expressions $\{a > 10, a + b, m * a\}$ may be used in obtaining the exit value of n. For S, we have $R_2(a > 10, m)$, $R_2(a + b, m)$, $R_2(a > 10, n)$, $R_2(a + b, n)$, $R_2(m * a, n)$.

$R_3(v_1, v_1)$ can be in interpreted as "the entry value of variable v_1 may be used in obtaining the exit value of variable v_2 in T". $R_3(v_1, v_2)$ implies that either: (i) the entry value of v_1 may be used in obtaining value of some expression e, which in turn may be used in obtaining the exit value of v_2 in T, or (ii) there is an assignment statement $v_2 = v_1$ that is preserved in T. An assignment statement $x = y$ is said to be *preserved* in T, if there exists a path in T that does not reassign x. R_3 can be expressed in terms of R_1 and R_2 as follows:
$R_3 = R_1R_2 \cup A$, where $A = \{(v_1, v_2)$, s.t there is an assignment statement $v_2 = v_1$ that is preserved in $T\}$.

In example S, $R_1R_2 = \{(a, m), (a, n), (b, m), (b, n)\}$, $A = \{(b, k)\}$, and

$R_3 = R_1 R_2 \cup A = \{(a,m),(a,n),(b,m),(b,n),(b,k)\}$.

5.5.2 Construction of Information-Flow Relations

The information-flow relations R_1 and R_2 are constructed using *use-def* [19] and *def-use* [19] analyses. *Use-def* analysis is used to create "use-definition chains" or "ud-chains". "Use-definition chains" are lists, for each use of a variable, of all the definitions that *reach* that use. We say a variable is *used* at statement s if its r-value may be required. For example, the statement: $(m = a + b)$, contains a use of a, a use of b and a definition of m. The sequence of statements:
$a = 5; m = c + d; m = a + b; p = m + n; q = a + t;$
contains one use and two definitions of m. Definition d_1: $(m = c + d)$ is overwritten by definition d_2:$(m = a + b)$. We say that definition d_2 *reaches* the use of m in statement $(p = m + n)$. Similarly, definition d_3:$(a = 5)$ reaches the use of a in statement $(m = a + b)$. We have $ud(m, p = m + n) = \{d_2\}$, and $ud(a, m = a + b) = \{d_3\}$.

Def-use analysis is used to create "definition-use chains" or "du-chains". The du-chaining problem is to compute for a definition d of variable x, the set of uses s of x such that there is a path from d to s that does not redefine x. In the sequence of statements:
$a = 5; m = c + d; m = a + b; p = m + n$
definition d_2:$(m = a + b)$ reaches the use of m in $(p = m + n)$. Similarly, definition d_3:$(a = 5)$ reaches the use of a in statement $(m = a + b)$, and the use of a in statement $(q = a + t)$. We have $du(d_2) = \{(m, p = m + n)\}$, and $du(d_3) = \{(a, m = a + b), (a, q = a + t)\}$.

By taking transitive closures of du-chains and ud-chains, relations R_1 and R_2 are constructed. From R_1, R_2, and A relation R_3 is constructed. Matrix M is constructed from R_3 for input and output variables. While constructing M, the *path* information is also taken into account (i.e. which expression or assignment statement was responsible in establishing the dependency between a particular output and input variable) as provided by relations R_1 and R_2. For simplicity sake, readers can assume that M stores R_3 for input and output variables (i.e $M = R_3(P, O)$). In Section 5.6, we explain how M can slightly differ from R_3 for certain programs.

```
int f(byte x1, byte y1, byte x2, byte y2, int k){
       byte x, y, dist, avg_x, avg_y;
       x = (x2 - x1)^2;
       y = (y2 - y1)^2;
       dist = sqrt(x + y);
       output(dist);
       if(k > 100){
           avg_x = (x1 + x2)/2;
           avg_y = (y1 + y2)/2;
           output(avg_x);
           output(avg_y);
      }
}
```

Figure 5.2: *Function that Calculates Distance Between Two Cars and Average Values of Coordinates*

The data-flow analysis is conservative, as *may* relations are used instead of *must* relations. Figure 5.2 is the example of a function that calculates the exact distance between two cars. Variables $x1, y1, x2, y2$ are private input variables, k is a public input variable, $dist, avg\_x, avg\_y$ are output variables. (x_1, y_1) is the location of the first car and (x_2, y_2) is the location of the second car. The function also computes the average value of coordinates if the value of k is greater than 100. The information-flow relations for this function and the matrix M are given in Figure 5.3. Note that $R_3 = R_1 R_2 \cup A$ and $M = R_3(P, O)$.

5.5.3 Solving Information-Flow Relations

Given information-flow relations, how do we decide if the program satisfies non-inference? The proof in Section 5.3 shows that although non-inference is decidable in the case of independent program executions, it is not clear if a *generic* polynomial-time solution is possible. We solve this problem in context of location privacy, where it is reasonable to assume that all input and output variables are 8-bit integers. In our solution, information-flow relations as treated as linear equations. *It can be shown that a program satisfies non-inference, if the system of linear equations given by:* $M^T P = O$ *is unsolvable.* P is the set of private input variables, O is the set of output variables and M^T is the transpose of matrix M which stores $R_3(P, O)$.

$P = \{x1, y1, x2, y2\}, O = \{dist, avg\_x, avg\_y\}$
$V = \{x1, y1, x2, y2, k, x, y, dist, avg\_x, avg\_y\}$
$E = \{(x2-x1)^2, (y2-y1)^2, sqrt(x+y), (dist > k), (x1+x2)/2, (y1+y2)/2\}$
$A = \{I\}$, I represents Identity: $(v = v)$

$$R_1 = V \times E = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$R_2 = E \times V = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_3 = V \times V = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M = P \times O = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

Figure 5.3: *Information-flow Relations for the Distance Example*

A system of linear equations given by $AX = B$, has a solution only if : rank(A) = rank($[AB]$) = N, where A is M-by-N, X is N-by-1 and B is M-by-1. The notation $[AB]$ means that B is appended to A as an additional column. X is the matrix of unknowns, A is the coefficient matrix, and B is the right-hand-side matrix. The rank of a matrix denotes the number of independent rows in the matrix and hence, the number of independent equations in the set. To get a solution, the rank of the coefficient matrix (A should be equal to the number of unknowns. If all the equations in the set are assumed to be independent, it suffices to check if $N = M$.

We represent dependency between private input variables P and output variables O as a set of linear equations: $M^T P = O$ (as shown in Figure 5.4). The intuition behind this representation is as following: let $p_1, p_2, .., p_k \in P$ be the set of private

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x1 \\ y1 \\ x2 \\ y2 \end{bmatrix} = \begin{bmatrix} dist \\ avg\_x \\ avg\_y \end{bmatrix}$$

$Rank(M^T) = 3 < (|P| = 4)$

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x1 \\ y1 \\ x2 \\ y2 \end{bmatrix} = \begin{bmatrix} dist \\ avg\_x \end{bmatrix}$$

$Rank(M_1^T) = 2 < (|P| = 4)$

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x1 \\ y1 \\ x2 \\ y2 \end{bmatrix} = \begin{bmatrix} avg\_x \\ avg\_y \end{bmatrix}$$

$Rank(M_2^T) = 2 < (|P| = 4)$

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x1 \\ y1 \\ x2 \\ y2 \end{bmatrix} = \begin{bmatrix} dist \\ avg\_y \end{bmatrix}$$

$Rank(M_3^T) = 2 < (|P| = 4)$

$$\begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x1 \\ y1 \\ x2 \\ y2 \end{bmatrix} = \begin{bmatrix} dist \end{bmatrix}$$

$Rank(M_4^T) = 1 < (|P| = 4)$

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \end{bmatrix} = \begin{bmatrix} avg\_x \end{bmatrix}$$

$Rank(M_5^T) = 1 < (|P_1| = 2)$

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} y1 \\ y2 \end{bmatrix} = \begin{bmatrix} avg\_y \end{bmatrix}$$

$Rank(M_6^T) = 1 < (|P_2| = 2)$

Figure 5.4: *Linear Equations for the Distance Example*

input variables for which, $M(p_i, o) = 1$, then it can be said that $o = f(p_1, p_2, ..p_k)$. In other words, an output variable can be represented as a function of all the private input variables that may affect its value. Public input variables are treated as constants, as they are known to the adversary. The simplest representation of a function is a linear equation. In reality, the function may be a higher-order equation involving complex operations. By adopting linear equation as the representation of all the functions, we guarantee a conservative analysis. If the system of linear equations cannot be solved, then values of private input variables (which are the unknowns in these equations) cannot be inferred from the values of output variables. However, since our analysis is conservative, we may have false negatives. That is, if a system of equations can be solved then it does not necessarily mean that the program violates non-inference. But the analysis will reject such a program.

We assume that all the equations are independent. Let R be the number of rows of M^T and C be the number of columns. We check if $R < C$. We carry out this check for all the matrices that can be derived from M^T by deleting one or more rows of M^T. This is because we want to know if the value of *any* input variable can be inferred from output variables. In other words, we want to know if any subset of the linear equations can be solved. If the check is satisfied for M^T and all its sub-matrices, the program satisfies non-inference.

Figure 5.4 shows the system of equations $M^T P = O$ for the example program of Figure 5.2, and all the subsystems that can be derived from M^T. The check $R < C$ is satisfied by all the subsystems. This implies that the set of linear equations is unsolvable. The example program therefore satisfies non-inference.

5.6 Implementation and Evaluation

We have implemented a system that decides non-inference for Java programs. This system has two main components. The first component carries out a static analysis of a given program and derives information-flow relations. The second component translates the information-flow relations into a set of a linear equations and inspects solvability. The implementation was done using Soot 2.2.1 [10] and Indus 0.7 [7]. Soot provides an API for analyzing and instrumenting Java bytecode. Indus provides an API for data-flow analysis. The current implementation does not support inter-procedural analysis, and assumes that input and output variables are 8-bit integers.

In absence of real applications that make use of location, the analyzer was evaluated on a self-written benchmark of Java programs. The benchmark consists of simple location based applications such as *Distance* (which returns distance between two cars), *Density* (which returns number of vehicles in a given region), *Speed* (which returns average speed of cars in a given region), *Average* (which returns average value x and y coordinates of the vehicles in the database). The benchmark also includes some standard applications and attacks, such as *PasswordChecker* [72, 53], *Wallet* [72], *WalletAttack* [72], *AverageAttack* [72]. These are in addition to the several microscopic Java

programs that were used in the testing of the implementation.

The evaluation tries to answer the following questions:

- How bad are false-negatives?
- Can there be false-positives?
- What is the average running time of the analysis?

Case Study 1: *AverageAttack [72]*

Suppose x_1,..x_n stores the x-coordinates of n vehicles (which is private). The average x-coordinate computation is intended to release the average but no other information about x_1,..x_n:

Average $= (x_1 + .. + x_n)/n$; output($Average$)

It is possible to formulate a laundering attack on the *average* program that leaks the x-coordinate of vehicle i:

$x_1 = x_i$;..$x_n = x_i$;

Average $= (x_1 + .. + x_n)/n$; output($Average$)

The system of linear equations for *AverageAttack* given by $M^T P = O$ is:

$$\begin{bmatrix} 0 & .. & 0 & 1 & 0 & .. & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ .. \\ x_i \\ .. \\ x_n \end{bmatrix} = \begin{bmatrix} Average \end{bmatrix}$$

which is reduced to:

$$\begin{bmatrix} 1 \end{bmatrix} \begin{bmatrix} x_i \end{bmatrix} = \begin{bmatrix} Average \end{bmatrix}$$

$P_1 = \{x_i\}$, $Rank(M^T) = 1 = (|P_1| = 1)$

This system of equations is solvable. Therefore, *AverageAttack* does not satisfy non-inference and is rejected by our analyzer.

Case Study 2: *Wallet and WalletAttack [72]*

Consider an electronic shopping scenario. Suppose p stores the amount of money in a customer's electronic wallet (which is private), q stores the amount of money already spent (which is public), and c stores the cost of item to be purchased (which is public). The following code snippet(*Wallet*) checks if the amount of money in the wallet is sufficient and, if so, transfers the amount c from the wallet to the spent-so-far variable q, and outputs q:

if $(p > c)$ then $(p = p - c; q = q + c)$; output(q)

The system of linear equations for *Wallet* given by $M^T P = O$ is:

$$\begin{bmatrix} 1 \end{bmatrix} \begin{bmatrix} p \end{bmatrix} = \begin{bmatrix} q \end{bmatrix}$$

$$Rank(M^T) = 1 = (|P| = 1)$$

This system of linear equations is solvable. Therefore, *Wallet* is rejected by our analyzer. However, it is easy to see that *Wallet* satisfies non-inference as the adversary cannot learn the value of p from q. This is an example of an application where our analyzer reports a false-negative. The reason for this is that the expression $(p > c)$ *may* affect the value of q in statement $q = q + c$. Therefore, $R_2(p > c, q) = 1$. We have $R_1(p, p \geq c) = 1$. From here, $R_3(p, q) = 1$. Therefore our analysis assumes dependency between variables q and p. There is indeed an *implicit* flow of information from p to q. It is safer to assume that the adversary may be able to infer the value of q from p, although in this particular example it cannot. In general: *implicit information-flows* may result in false-negatives.

Now, we give an example to show the importance to having an analysis that is *conservative* and may occasionally report false-negatives. Consider the following code snippet (*WalletAttack*):

```
n = length(p)
while(n >= 0){
```

```
  c = 2^{n - 1}
  if(p > c){
    p = p - c;
    q = q + c;
    n = n - 1;
 }
}
output(q)
```

This code snippet leaks the value of p bit-by-bit to q. Our analyzer is able to detect it. The system of linear equations for *WalletAttack* given by $M^T P = O$ is:

$$\begin{bmatrix} 1 \end{bmatrix} \begin{bmatrix} p \end{bmatrix} = \begin{bmatrix} q \end{bmatrix}$$

$$Rank(M^T) = 1 = (|P| = 1)$$

This system of linear equations is solvable. *WalletAttack* is, therefore, rejected by our analyzer.

Case Study 3: *PasswordChecker [53, 72]*

Consider UNIX-style password checking where the system database stores hashes of password-salt pairs. *Salt* is a publicly readable string stored in the database for each user id, as a protection against dictionary attacks. For a successful login, a user is required to provide a password such that the hash of the password and salt matches the hash from the database. The following code snippet captures this functionality:

```
byte check(byte username, byte password){
  byte match =0;
  for(i = 0; i < database.length; i++){
    if(hash(username, password)
       == hash(salts[i], passwords[i])){
       match = 1;
       break;
```

```
    }
  }
  output(match);
}
```

This commonly used program is rejected by non-interference, as the value of public boolean variable *match* depends on private variables. It is easy to see that this program satisfies non-inference(as the username/password cannot be inferred from the binary output). Our analyzer accepts this program.

Case Study 4: *IfAttack*

Consider the following code snippet:

```
void fun(byte a, byte b, int i){
    byte out = 0;
    if(i >= 1){
      out = a;
    }else{
      out = a + b;
    }
    output(out)
}
```

Variables a and b are private input variables, variable i is a public input variable. If $i < 1$ then the output variable *out* leaks the value of input variable a, otherwise not. This program does not satisfy non-inference. Our analyzer is able to detect this. This is because, while constructing matrix M from R_3, we use path information. For this code snippet,

$$R_3 = P \times O = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, M = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

The system of linear equations $M^T P = O$, is:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} out \\ out \end{bmatrix}$$

Note that we have two equations corresponding to the two paths. Path information is available from relations R_1 and R_2. A subsystem of this system is given by:

$$\begin{bmatrix} 1 \end{bmatrix} \begin{bmatrix} a \end{bmatrix} = \begin{bmatrix} out \end{bmatrix}$$

which is solvable, thereby violating non-inference.

5.6.1 Discussion

PasswordChecker is a simple example of a commonly-used application that does not satisfy non-interference but satisfies non-inference. *Distance*(which calculates distances between two cars) is a function that satisfies non-inference, and whose quality would suffer if spatial/temporal cloaking is used. As described before, Geographical Routing Service can make use of *Distance*. Similarly, functions such as *Density*(which calculates density of vehicles in a region) or *Speed*(which calculates average speed of vehicles in a region) satisfy non-inference and can benefit from our framework. Traffic survey applications can make use of these functions. All these functions have been tested with our analyzer. These are all examples of code-snippets that compute a result based on location information, and will be part of bigger applications that would run on the LBS side (such as Geographical Routing Service, or Traffic Information Service). Many of these applications can work with temporal/spatial cloaking as well, with compromised quality of service. While Spatial/temporal cloaking is more generally applicable, our framework is more suitable for applications that need fine grained location information. Through this framework, we provide a choice. We do not believe that the solution in this paper can be a replacement for spatial/temporal cloaking. Non-inference is applicable to sensor data privacy in general. Location is just an example of time-series information coming from location sensors.

How bad are false-negatives? Through the *Wallet* example, we showed that false-negatives are possible, as our analysis is conservative. In general, *implicit* information-flows can lead to false-negatives being reported. At the same time, through the *WalletAttack* example, we showed why it is better to have a conservative analysis that occasionally reports false-negatives, as opposed to one that does not and can be attacked. Most of the applications that satisfy non-inference but are rejected by our analyzer, can be rewritten with minor syntactic modification to satisfy our analyzer.

Can there be false positives? We gave examples of a few well-known attacks (e.g *AverageAttack*, *WalletAttack*) that can break non-interference with declassification. These attacks are detected by our analyzer and rejected. Theoretically, we can show that it is not possible to launch attacks against our analyzer. Here, we sketch the proof idea: let $p_1, p_2, .., p_k \in P$ be the set of private input variables for which, $M(p_i, o) = 1$, then it can be said that $o = f(p_1, p_2, ..p_k)$. In other words, an output variable can be represented as a function of all the private input variables that may affect its value. Public input variables are treated as constants, as they are known to the adversary. The simplest representation of a function is a linear equation. In reality, the function may be a higher-order equation involving complex operations. By adopting linear equation as the representation of all the functions, we guarantee a conservative analysis. It can therefore be said that if a system of linear equations cannot be solved, then values of private input variables (which are the unknowns in these equations) cannot be inferred from the values of output variables. However, the converse does not hold.

What is the average running time of our analysis? The experiments were carried out on an IBM ThinkPad R51 with 1.5GHz Intel processor and 256MB RAM, running the Linux operating system. For the benchmark consisting of 11 Java programs, the running time of our analyzer ranges between 3.6 and 4.2 seconds (as shown in Figure 5.5). Typically, the untrusted code would be some kind of an aggregation function that would be at most a few hundred lines of code. Our method of constructing information-flow relations has a worst case time complexity of $O(|E|^2 \times |V|^3)$ (where E is the set of expressions and V is the set of variables). The worst case time complexity of deciding solvability of information-flow relations is $O(|O| \times |P|^3)$ (where $O \in V$ is

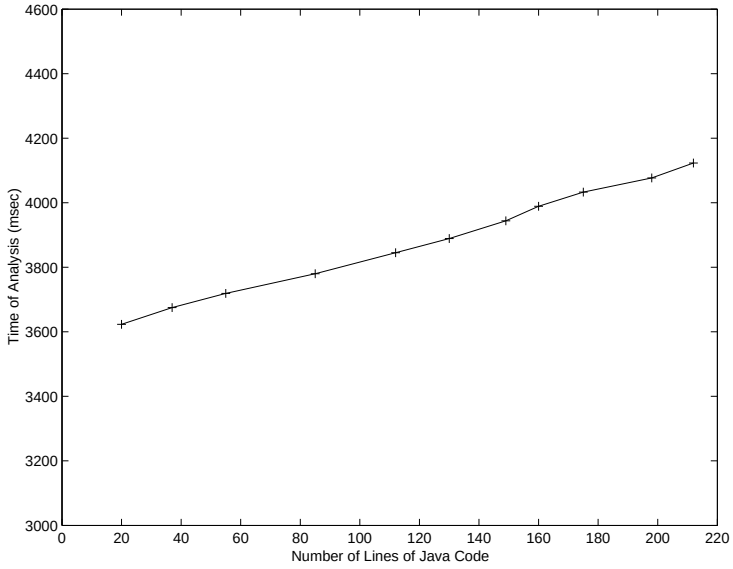

Figure 5.5: *Running Time of the Static Analyzer*

the set of output variables and $P \in V$ is the set of private input variables). The total worst case time complexity of our analysis is $O((|E|^2 + |O|) \times |V|^3)$. $|V|$ and $|E|$ would be directly proportional to the code size. Figure 5.5 shows that the time of analysis increases with the size of code. There is a constant load time for the analyzer which is around 3500msec.

5.6.2 Limitations

Through this information flow analysis, we are able to successfully defend against the common class of attacks, thereby raising the bar for the adversary. Currently, no guarantees are provided on how many bits of a private input variable can be inferred from output variables. The adversary can get some partial information about the location by crafting sophisticated attacks. Partial information leakage, however, is a less serious concern for location privacy. It can be argued that hiding even one bit of location information can guarantee a high degree of uncertainty, as the adversary would not be able to distinguish a vehicle from other vehicles around it (similar to spatial cloaking in k-anonymity). The verification tool can be combined with manual analysis to prevent even partial information leakage. The tool would then assist the manual

analyzer in identifying information leakage attacks, in the same way as a debugger assists the application developer in writing robust code.

In order to make the analysis more robust, it should be combined with abstract execution. Abstract execution can determine the dependency between two variables more accurately than static data flow anlysis. It can establish whether or not a variable *will certainly* affect the value of another variable at runtime, and if so, to what extent. The main idea is to execute the program in a sandbox and perturb the value of private input variables, one at a time, to see how public output variables are affected.

5.7 Summary

Location-aware personal computing requires location information of the user to be shared with untrusted services which can cause potential privacy breach. In this chapter, we presented a new information-flow control model called non-inference for guarding location-privacy against untrusted services. Non-inference allows public data to be derived from private data but not vice versa. We discussed the theoretical implications of non-inference. We showed that non-inference is undecidable in general, but decidable for applications where multiple executions are independent such as location-based services. We showed that it can be enforced *conservatively* using static program analysis. The main idea is to apply the theory of solvability of linear equations to information-flow relations derived by data-flow analysis. We implemented a system that decides non-inference for Java programs using static analysis, and tested it on a benchmark and case studies known in the literature.

Chapter 6

Summary and Conclusions

In this dissertation, we pointed out the two key hurdles that prohibit the widescale adoption of *pervasive computing*: (1) limited cognitive bandwidth, and (2) dependency on a ubiquitous computing infrastructure. We proposed location-aware personal computing as an way of getting close to the pervasive computing vision. At the heart of location-aware personal computing is the use of smart phones as computing devices, and user location as a means of discovering and personalizing services. This dissertation investigates the various issues in location-aware personal computing, namely: (1). discovering and provisioning locally embedded services to the smart phone of the user without pre-configuration, (2). determining user location without requiring extra infrastructure, (3). managing the battery lifetime of the smart phone and (4). protecting location information of the user from illegal access and misuse. The conclusions of this research are spread over all the four research directions, which we summarize in what follows. At a higher level, the main conclusion of this research is that the cooperative use of smart phones and user location can be instrumental in devising low-cost, easy-to-use and non-intrusive solutions for pervasive computing without requiring significant infrastructure support.

It is not always possible to customize web services to serve as local services, therefore, some amount of computing would have to be embedded in physical space in order to realize the vision of ubiquitous computing. Protocols are needed for seamlessly discovering these locally embedded services and provisioning them to the user. Several service discovery protocols have been proposed in the past (see Table 1.1). These protocols can be roughly classified into two categories: *client-service* model and *client-service-directory* model. In *client-service* model, clients directly query the services. In

client-service-directory model, clients query directories that cache service information.

While the *client-service* model requires less infrastructure and is more suitable for nomadic environments, it is not nearly as powerful as the *client-service-directory* model. In location-aware mobile computing, smart phones serve as clients. Bluetooth SDP has been incorporated in smart phones and follows the *client-service* discovery model. Since smart phones have always-on internet connectivity in the form of 3G/GPRS, directories can be conveniently maintained on the internet.

This dissertation presents a service discovery, interaction and payment protocol (SDIPP) [68] that follows the *client-service-discovery* model. SDIPP is smart phone centric; it exploits *dual-connectivity* (Bluetooth and GPRS) on smart phones to simultaneously connect with local services and web services. SDIPP takes a hybrid approach to discovery in order to make the discovery process as generic as possible. In the first phase, the services are discovered. In the second phase, a service is selected (based on the location of the user) and a mechanism for interacting with the service is decided. In the third and final phase, the user interacts with the service and pays for its usage (when required). The discovery component is hierarchical and opportunistic– the network interfaces of the phone as well as those of the neighbouring devices are utilized to discover services. Multi-hop discovery is carried out using Portable Smart Messages(SM) [61]. The code for interacting with services is discovered and downloaded on the fly, eliminating the need to have any prior knowledge of the services. The payment protocol is based on the idea of Millicent scrips [35] and is provably secure. Evaluation of SDIPP on Sony Ericsson P900 phones indicate satisfactory performance. Applications developed on top of this protocol and tested on Sony Ericsson P900 phones show the applicability of SDIPP.

In order to enable location-aware personal computing, continuous location updates are needed both outdoors and indoors. Although several solutions have been proposed for determining user location indoors, each of them relies on some infrastructure support in the environment, which increases the overhead of determining location and decreases the chances of deployment.

This dissertation presents two infrastructureless solutions [67, 64] for indoor localization, which can potentially be combined together for higher accuracies. In the camera-phone-based localization solution [67], images are periodically captured by the camera phone worn by the user as a pendant and transmitted to a web server over 3G. The web server maintains a database of images with their corresponding locations. Upon receiving an image, the web server compares it with stored images, and based on the match, estimates user's location. This is accomplished with off-the-shelf image matching algorithms, by tailoring them for our purpose. This solution does not require any infrastructure to be installed in the environment; neither custom hardware nor wireless access points are required; physical objects do not have to be "tagged" and users do not have to carry any special device. User location can be determined with room-level precision with more than 90% success probability, and corner-level precision with more than 80% success probability. User orientation is determined along with location. The main limitation of this approach is that it is not resilient to varying lighting conditions.

In the light-intensity-based localization solution [64], the location of the user is determined based on the intensity of light incident on a light sensor worn by the user. We show that every room has a unique light-intensity fingerprint, which can be used to identify it. This solution also does not require any infrastructure to be installed in the environment, and the user is only required to carry a tiny light sensor. Furthermore, it only takes around a minute to collect training data for a room. Experimental results show that this approach works very well under static lighting conditions (with upto 90% success probability), but moderately well in the presence of sunlight (with upto 68-73% success probability).

Both the localization solutions are infrastructureless and can be combined together to attain higher accuracies. The light sensor could either be embedded in the phone or iPod of the user or worn separately as a clip. We believe that a practical solution for localization would be a fusion system that incorporates an array of sensors (such as camera, light sensor, GPS, WiFi interface, accelerometer etc), derives location information from each of the sensors and fuses them together to obtain accurate results. Also,

each of the sensors could fail under certain conditions (for example the light sensor may not work very well when there is interference from sunlight) but the fusion system as a whole should be quite robust.

Location-aware personal computing is centered around the idea of running applications on resource constrained personal devices (e.g smart phone), many of which would run as background tasks including daemons for determining user's location, daemons for listening to incoming network requests, daemons for warming the phone cache etc. These applications would compete for the limited resources on the device. Battery lifetime is by far the most crucial resource. On smart phones, the interfaces that inform the user of the battery levels have not kept up with the evolution of the capabilities of these devices. A simple "battery remaining" or even "time remaining" does not enable the user to make the right decisions as to the spend of the energy budget and the request for recharging.

Location information has never been exploited in managing battery lifetime. This dissertation describes a location-aware battery management scheme [66, 63] for smart phones, which is based on two key observations. First, it is necessary to create an energy budget for different applications in order to prevent low priority applications (such as background tasks) from affecting the availability of high priority applications (such as telephony). Second, the knowledge of when the user will recharge the phone next is crucial to managing battery lifetime on phones. This is because the knowledge of when the next recharge will happen determines the total battery lifetime available to applications. User studies show that users typically charge their phones at fixed locations [25]. Hence, by predicting the whereabouts of the user, it is possible to predict charging opportunities. In this dissertation, we present a system architecture for logging user information (such as location, call duration, etc) on the phone and a set of algorithms for making predictions based on these logs. Experimental results show that these predictions can be made with fairly high accuracy. This allows the system to determine the amount of battery lifetime needed for safe execution of crucial applications and to warn the user if one or more non-crucial applications need to be terminated.

Finally, this dissertation investigates the problem of location privacy. Extensive deployment of location-aware computing endangers user's location privacy and exhibits significant potential for abuse. The US government realized the seriousness of the this problem and released the *Location Privacy Protection Act* [8] in 2001. Unless the users are secure about their location privacy, they would be reluctant to use location-aware applications. This hurdle would have to be overcome if location-aware computing is to be globally accepted.

This dissertation proposes a *service-specific* location privacy approach, centered on an information flow control analysis. Information-flow control policies tend to impose restrictions on the manner in which sensitive data flows through a program/system. The standard information-flow control model called *Non-interference* [59, 26] requires that any possible variation in private data must not cause a variation in public data. That is, if the value of a public variable q depends on that of a private variable p then non-interference is violated. In effect, non-interference isolates private data from public data. In doing so, it guarantees that the publicly observable behavior of a system/program that does not reveal *anything* about its private behavior. However, data isolation is an extreme measure, and in many real applications it is not possible to isolate private data from public data. This dissertation proposes a weaker model of information-flow control called *non-inference* [62]. *Non-inference* requires that the adversary should not be able to *infer* the value of a private variable based on the values of public variable. Non-inference, therefore, allows information to flow from private variables to public variables, but prohibits the adversary from learning the value of any private variable from public variables. In this dissertation, we show how non-inference can be used to preserve location privacy.

We discuss the theoretical implications of non-inference. We show that non-inference is undecidable in general, but decidable for applications where multiple executions are independent such as location-based services. We show that it can be enforced *conservatively* using static program analysis. The main idea is to apply the theory of solvability of linear equations to information-flow relations derived by data-flow analysis. We implemented a system that decides non-inference for location-based services using static

program analysis. Experimental results obtained on a benchmark indicate success.

6.1 Directions for Research

We are still far from the goal of "location for free". The localization mechanisms described in this dissertation, though infrastructureless, are not free from configuration effort, which can make deployment hard. We see two directions for research. One is to minimize the configuration effort required for localization. There has already been some research in this direction for WiFi-based localization systems [50]. The other is to create various combinations of the already proposed localization systems, in an effort to create a fusion system, and carry out comparative studies.

Simultaneously, we need to find ways of disclosing location information to the computing infrastructure "for free". This implies solving the complex problem of constraining flow of location information, such that user privacy is not breached. The solution described in this dissertation only applies to a certain class of applications. Different classes of applications will demand different solutions.

We need to start deploying applications that can benefit from location information. This is hard to accomplish at a global scale because we do not yet have "location for free". However, efforts can be launched at campus and university level.

Location is considered the most important context information. User activity is another example of useful context information. In general, use of personal information can enable new applications. There is value in coming up with ways of sensing, storing and analyzing personal information. Furthermore, combining dynamic personal information (e.g location, activity, mobile media) that can be sensed with body-worn sensors, with personal information stored on the web (such as on community websites) can enable novel and secure usage models.

6.2 The Symbian and iPhone Era

The personal computer is climbing off its desktop perch and hopping into the pockets of millions of people [16]. The adoption of Symbian OS [11] by mobile phone makers, such

as Sony Ericsson and Nokia, brought about the smart phone revolution in 2003-2004. In 2007, Apple Inc introduced the iPhone, which energized the industry and took the smart phone revolution to the next level. In the second-half of 2008, Google is expected to release Google Phone [16], which unlike its predecessors will be open source. This is just beginning of the making of the *computer for the twenty first century*.

6.3 In Retrospect

Pervasive computing is an application-driven field and as such is ill-defined and unstructured for the lack of definitions, algorithms and architectures. Research is primarily guided by visions and beliefs because it is hard to tell, in a scientific way, which systems/applications will penetrate the market. To realize the diverse set of applications, research has to be multi-disciplinary, which adds to the complexity. The exploratory and unstructured nature of pervasive computing research is, therefore, justified to some extent. Besides, this field is still in a nascent stage. Unlike networks/compilers/operating systems, which exist and are widely used and deployed, pervasive computing exists mostly in labs or in specialized environments or in the form of single isolated applications. As a result, progress is hard to measure and prove.

Success of pervasive computing would depend on the advances in evolving areas in computer science, namely vision, artificial intelligence, wireless communication and human-computer interaction. Simultaneously, we need to be on a quest for mobile applications that users would be willing to *pay for*. I believe that in the near future, these applications would either be smart phone centric or vehicle centric. Vehicular computing is a sister-field of pervasive computing that has found enormous support both from academia and industry in the recent past (applications include congestion control, safety messaging and highway-infotainment).

Users first, computers second. This mantra should never be forgotten. Pervasive computing is not so much an area of research as a goal. And this goal cannot be accomplished by solving the technical challenges alone. It requires a deep understanding of what users *really* want. This goes beyond the scope of computer science. And is

perhaps the biggest challenge.

Despite the many challenges that stand in the way of pervasive computing becoming a reality, I believe that it is just a matter of time before we see it happen.

References

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

[2] Mobile Media API(MMAPI), http://java.sun.com/products/mmapi/index.jsp.

[3] Reality Mining, http://reality.media.mit.edu/.

[4] "CyberCash." http://www.cybercash.com.

[5] "Digicash." http://www.digicash.com.

[6] "First Virtual Holdings Incorporated." http://www.fv.com.

[7] "Indus." http://indus.projects.cis.ksu.edu/.

[8] "Location privacy protection act, 2001." http://www.theorator.com/bills107/s1164.html.

[9] "OTA Provisioning." http://java.sun.com/products/midp/ OTAProvisioning-1.0.pdf.

[10] "Soot: A java optimization framework." http://www.sable.mcgill.ca/soot/.

[11] "Symbian OS." http://www.symbian.com/.

[12] "Universal Description, Discovery and Integration (UDDI)." http://www.uddi.org, 2000.

[13] "White Paper : Salutation Architecture," 1998. http://www.salutation.org/whitepaper/originalwp.pdf.

[14] "Bluetooth Specification Part E. Service Discovery Protocol (SDP)," 1999. http://www.bluetooth.com.

[15] "Universal Plug and Play Device Architecture," 1999. Microsoft Corporation. http://www.upnp.org.

[16] "Google enters the wireless world," November 2007. http://www.nytimes.com/2007/11/06/.

[17] L. Aalto, N. Gothlin, J. Korhonen, and T. Ojala, "Bluetooth and wap push based location-aware mobile advertising system," in *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, 2004.

[18] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley, "The design and implementation of an intentional naming system," in *Proceedings of the seventeenth ACM symposium on Operating systems principles*, 1999, pp. 186–201.

[19] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: principles, techniques, and tools*. Addison-Wesley Longman Publishing Co., Inc., 1986.

[20] M. Anand, E. B. Nightingale, and J. Flinn, "Ghosts in the machine: interfaces for better power management," in *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, 2004, pp. 23–35.

[21] B. Cox and J.D. Tygar and M. Sirbu, "Netbill security and transaction protocol," in *Proceedings of the First USENIX Workshop on Electronic Commerce*, 1995, pp. 6–6.

[22] P. Bahl and V. N. Padmanabhan, "RADAR: An in-building RF-based user location and tracking system," in *INFOCOM (2)*, March 2000, pp. 775–784.

[23] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H.-I. Yang, "The case for cyber foraging," in *Proceedings of the 10th workshop on ACM SIGOPS European workshop*, 2002, pp. 87–92.

[24] R. Ballagas, M. Rohs, J. G. Sheridan, and J. Borchers, "Sweep and point & shoot: Phonecam-based interactions for large public displays," in *CHI '05: Extended abstracts on Human factors and computing systems*, 2005, pp. 1200–1203.

[25] N. Banerjee, A. Rahmati, M. Corner, S. Rollins, and L. Zhong, "Users and batteries: Interactions and adaptive energy management in mobile systems," in *Ninth International Conference on Ubiquitous Computing*, 2007, pp. 217–234.

[26] D. Bell and L. LaPadula, "Secure computer system: Unified exposition and multics interpretation," *Technical Report ESD-TR-75-306, Electronics Systems Division, Mitre Corp.*, 1975.

[27] G. Borriello, A. Liu, T. Offer, C. Palistrant, and R. Sharp, "Walrus: wireless acoustic location with room-level resolution using ultrasound," in *MobiSys '05: Proceedings of the 3rd international conference on Mobile systems, applications, and services*, 2005, pp. 191–203.

[28] R. Bruno and F. Delmastro, "Design and analysis of a bluetooth-based indoor localization system," *PersonalWireless Communications*, pp. 711–725, 2003.

[29] D. E. Denning, "A lattice model of secure information flow," *Commun. of the ACM*, vol. 19, no. 5, pp. 236–243, 1976.

[30] D. E. Denning and P. J. Denning, "Certification of programs for secure information flow," *Commun. of the ACM*, vol. 20, no. 7, pp. 504–513, 1977.

[31] S. Duri, M. Gruteser, X. Liu, P. Moskowitz, R. Perez, M. Singh, and J.-M. Tang, "Framework for security and privacy in automotive telematics," in *WMC '02: Proceedings of the 2nd international workshop on Mobile commerce*, 2002, pp. 25–32.

[32] J. Flinn and M. Satyanarayanan, "Energy-aware adaptation for mobile applications," in *SOSP '99: Proceedings of the seventeenth ACM symposium on Operating systems principles*, 1999, pp. 48–63.

[33] S. Funfrocken, "Transparent Migration of Java-Based Mobile Agents," in *Mobile Agents*, 1998, pp. 26–37.

[34] R. Giacobazzi and I. Mastroeni, "Abstract non-interference: parameterizing non-interference by abstract interpretation," in *POPL '04: Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 2004, pp. 186–197.

[35] S. Glassman, M. Manasse, M. Abadi, P. Gauthier, and P. Sobalvarro, "The Millicent protocol for inexpensive electronic commerce," in *Proceedings of the 4th World Wide Web Conference* , 1995.

[36] M. Gruteser and D. Grunwald, "Anonymous usage of location-based services through spatial and temporal cloaking," in *MobiSys*, 2003, pp. 31–42.

[37] E. Guttman, "Service location protocol: Automatic discovery of ip network services," *IEEE Internet Computing*, vol. 3, no. 4, 1999.

[38] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman, "Protection in operating systems," *Commun. ACM*, vol. 19, no. 8, pp. 461–471, 1976.

[39] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster, "The anatomy of a context-aware application," in *Mobile Computing and Networking*, 1999, pp. 59–68.

[40] T. Heath, E. Pinheiro, J. Hom, U. Kremer, and R. Bianchini, "Code transformations for energy-efficient device management," in *IEEE Transactions on Computers*, volume 53, 2004, pp. 974–987.

[41] L. Iftode, C. Borcea, N. Ravi, P. Kang, and P. Zhou, "Smart phone: An embedded system for universal interactions," in *Proceedings of the tenth International Workshop on Future Trends in Distributed Computing Systems*, 2004, pp. 88–94.

[42] C. E. Jacobs, A. Finkelstein, and D. H. Salesin, "Fast multiresolution image querying," in *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, 1995, pp. 277–286.

[43] X. Jiang and J. A. Landay, "Modeling privacy control in context-aware systems," *IEEE Pervasive Computing*, vol. 1, no. 3, pp. 59–63, 2002.

[44] P. Kang, C. Borcea, G. Xu, A. Saxena, U. Kremer, and L. Iftode, "Smart Messages: A Distributed Computing Platform for Networks of Embedded Systems," *The Computer Journal*, vol. 47, no. 4, pp. 475–494, 2004.

[45] T. Kato, T. Kurita, N. Otsu, and K. Hirata, "A sketch retrieval method for full color image database," in *In Proceedings of International Conference on Pattern Recognition*, 1992, pp. 530–533.

[46] U. Kremer, J. Hicks, and J. Rehg, "A compilation framework for power and energy management on mobile computers," in *Proceedings of the 14th International Workshop on Parallel Computing (LCPC'01)*, 2001, pp. 115–131.

[47] J. Krumm, S. Harris, B. Meyers, B. Brumitt, M. Hale, and S. Shafer, "Multi-camera multi-person tracking for easyliving," in *VS '00: Proceedings of the Third IEEE International Workshop on Visual Surveillance (VS'2000)*, 2000, pp. 3–10.

[48] A. LaMarca, Y. Chawathe, S. Consolvo, J. Hightower, I. Smith, J. Scott, T. Sohn, J. Howard, J. Hughes, F. Potter, J. Tabert, P. Powledge, G. Borriello, and B. Schilit, "Place lab: Device positioning using radio beacons in the wild," in *Proceedings of the Third International Conference on Pervasive Computing*, 2005, pp. 116–133.

[49] P. Li and S. Zdancewic, "Downgrading policies and relaxed noninterference," *SIGPLAN Not.*, vol. 40, no. 1, pp. 158–170, 2005.

[50] H. Lim, L.-C. Kung, J. C. Hou, and H. Luo, "Zero-configuration, robust indoor localization: Theory and experimentation," in *Proceedings of 25th Conference on Computer Communications(INFOCOM)*, April 2006, pp. 1–12.

[51] A. C. Myers, "JFlow: Practical mostly-static information flow control," in *Symposium on Principles of Programming Languages*, 1999, pp. 228–241.

[52] A. C. Myers and B. Liskov, "Protecting privacy using the decentralized label model," *ACM Transactions on Software Engineering and Methodology*, vol. 9, no. 4, pp. 410–442, 2000.

[53] A. C. Myers, A. Sabelfeld, and S. Zdancewic, "Enforcing robust declassification," in *CSFW '04: Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW'04)*, 2004, pp. 172–186.

[54] W. Niblack, R. Barber, and et al, "The qbic project: querying images by content using color, texture and shape," in *In Proceedings of SPIE Storage and Retrieval for Image and Video Databases*, 1993, pp. 173–187.

[55] R. J. Orr and G. D. Abowd, "The smart floor: a mechanism for natural user identification and tracking," in *CHI '00: CHI '00 extended abstracts on Human factors in computing systems*, 2000.

[56] V. Otsason, A. Varshavsky, A. LaMarca, and E. da Lara, "Accurate gsm indoor localization," in *Proceedings of the Seventh International Conference on Ubiquitous Computing*, 2005, pp. 141–158.

[57] S. N. Patel, K. N. Truong, and G. D. Abowd, "Powerline positioning: A practical sub-room-level indoor location system for domestic use.," in *Ubicomp*, 2006, pp. 441–458.

[58] A. D. Pierro, C. Hankin, and H. Wiklicky, "Approximate non-interference," in *CSFW '02: Proceedings of the 15th IEEE Computer Security Foundations Workshop (CSFW'02)*, 2002, pp. 3–17.

[59] S. policies and security models, "J.a. goguen and j. meseguer," in *In Proc. IEEE Symposium on Security and Privacy*, 1982, pp. 11–20.

[60] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan, "The cricket location-support system," in *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, 2000, pp. 32–43.

[61] N. Ravi, C. Borcea, P. Kang, and L. Iftode, "Portable Smart Messages for Ubiquitous Java-enabled Devices," in *First International Conference on Mobile and Ubiquitous Computing*, 2004, pp. 412–421.

[62] N. Ravi, M. Gruteser, and L. Iftode, "Non-inference: A novel information flow control for location-based services," in *Third Annual International Conference on Mobile and Ubiquitous Systems: Networks and Services (Mobiquitous)*, July 2006, pp. 1–10.

[63] N. Ravi, L. Han, J. Scott, and L. Iftode, "Context-aware battery management for mobile phones," in *Proceedings of the Sixth International Conference on Pervasive Computing and Communications(PerCom)*, March 2008.

[64] N. Ravi and L. Iftode, "Fiatlux: Fingerprinting rooms using light intensity," in *Adjunct Proceedings of the Fifth Internationa; Conference on Pervasive Computing(Pervasive)*, May 2007.

[65] N. Ravi and L. Iftode, "Outdoor distributed computing with split smart messages," in *Lecture Notes in Computer Science*, volume 4322, June 2007, pp. 161–183.

[66] N. Ravi, J. Scott, and L. Iftode, "Towards context-aware battery management for mobile phones," in *Proceedings of the Fifth International Conference on Pervasive Computing(Pervasive)*, May 2007.

[67] N. Ravi, P. Shankar, A. Frankel, A. Algammal, and L. Iftode, "Indoor localization using camera phones," in *Proceedings of the Seventh IEEE Workshop on Mobile Computing Systems and Applications(WMCSA/HotMobile)*, 2006, pp. 1–7.

[68] N. Ravi, P. Stern, N. Desai, and L. Iftode, "Accessing ubiquitous services using smart phones," in *PERCOM '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*, 2005, pp. 383–393.

[69] M. Rohs, "Real-world interaction with camera-phones," in *Proceedings of the 2nd International Symposium on Ubiquitous Computing Systems*, 2004, pp. 74–89.

[70] M. Rohs and P. Zweifel, "A conceptual framework for camera phone-based interaction techniques," in *Proceedings of the Third International Conference on Pervasive Computing*, 2005, pp. 171–189.

[71] P. Ryan, J. McLean, J. Millen, and V. Gligor, "Non-interference, who needs it?," in *In Proceedings of the 14th IEEE Computer Security Foundations Workshop*, 2001, pp. 237–238.

[72] A. Sabelfeld and A. Myers, "A model for delimited information release," 2003.

[73] T. Sakamoto, T. Sekiguchi, and A. Yonezawa, "Bytecode Transformation for Portable Thread Migration in Java," in *ASA/MA*, 2000, pp. 16–28.

[74] D. Scott, R. Sharp, A. Madhavapeddy, and E. Upton, "Using visual tags to bypass bluetooth device discovery," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 9, no. 1, pp. 41–53, 2005.

[75] J. Scott and B. Dragovic, "Audio location: Accurate low-cost location sensing," in *Proceedings of the Third International Conference on Pervasive Computing*, 2005, pp. 1–18.

[76] M. J. Swain and D. H. Ballard, "Color indexing," *Int. J. Comput. Vision*, vol. 7, no. 1, pp. 11–32, 1991.

[77] L. Sweeney, "Achieving k-anonymity privacy protection using generalization and suppression," *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 10, no. 5, pp. 571–588, 2002.

[78] E. Toye, R. Sharp, A. Madhavapeddy, and D. Scott, "Using smart phones to access site-specific services," *IEEE Pervasive Computing*, vol. 4, no. 2, pp. 60–66, 2005.

[79] E. Truyen, B. Robben, B. Vanhaute, T. Coninx, W. Joosen, and P. Verbaeten, "Portable Support for Transparent Thread Migration in Java," in *ASA/MA*, 2000, pp. 29–43.

[80] A. Vellaikal and C. Kuo, "Content-based retrieval using multiresolution histogram representation," *Digital Image Storage Archiving Systems*, vol. 2602, pp. 312–323, 1995.

[81] J. Waldo, "The jini architecture for network-centric computing," *Communications of the ACM*, vol. 42, no. 7, pp. 76–82, 1999.

[82] R. Want, A. Hopper, V. Falco, and J. Gibbons, "The active badge location system," *ACM Transactions on Information Systems*, vol. 10, pp. 91–102, 1992.

[83] M. Weiser, "The computer for the twenty-first century," *Scientific American*, vol. 265, pp. 94–104, September 1991.

[84] A. Woodruff, K. Anderson, S. Mainwaring, and R. Aipperspach, "Portable, but not mobile: A study of wireless laptops in the home," in *Proceedings of the Fifth International Conference on Pervasive Computing(Pervasive)*, May 2007, pp. 216–233.

[85] S. Zdancewic, L. Zheng, N. Nystrom, and A. C. Myers, "Untrusted hosts and confidentiality: Secure program partitioning," in *Symposium on Operating Systems Principles*, 2001.

[86] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat, "Ecosystem: managing energy as a first class operating system resource," in *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, 2002, pp. 123–132.

Vita

Nishkam Ravi

Education

- Ph.D Computer Science, Rutgers University, New Jersey (January 2008)
- B.Tech Computer Science, Indian Insititute of Technology(IIT), Bombay, India (July 2002)

Occupations and Positions Held

- Summer Intern, IBM Watson Research Center (Summer 2005)

Publications

- Nishkam Ravi, James Scott, Lu Han and Liviu Iftode. Context-aware Battery Management for Mobile Phones. *Proceedings of the Sixth International Conference on Pervasive Computing and Communications(PerCom), Hong Kong, March 2008*
- Nishkam Ravi, Chandra Narayanawami, Mandayam Raghunath and Marcel Rosu. Securing Pocket Hard Drives. *IEEE Pervasive Computing (Special Issue on Security and Privacy), Volume 6, Number 4, Oct 2007*
- Nishkam Ravi, Stephen Smaldone, Liviu Iftode and Mario Gerla. Intelligent Lane Reservation for Highways (Position Paper). *Proceedings of the Tenth International IEEE Conference on Intelligent Transportation Systems, Seattle, WA, Sept 2007*
- Nishkam Ravi and Liviu Iftode. Outdoor Distributed Computing with Split Smart Messages. *Lecture Notes in Computer Science, Volume 4322, June 2007*
- Nishkam Ravi and Liviu Iftode. FiatLux: Fingerprinting Rooms Using Light Intensity. *Adjunct Proceedings of the Fifth International Conference on Pervasive Computing (Pervasive), Toronto, Canada, May 2007*
- Nishkam Ravi, James Scott and Liviu Iftode. Towards Context-aware Battery Management for Mobile Phones. *Adjunct Proceedings of the Fifth International Conference on Pervasive Computing (Pervasive), Toronto, Canada, May 2007*
- Fabio Picconi, Nishkam Ravi, Marco Gruteser and Liviu Iftode. Probabilistic Validation of Aggregated Data in Vehicular Ad-hoc Networks. *Proceedings of the Third ACM International Workshop on Vehicular Ad-hoc Networks (VANET), Los Angeles, CA, Sept 2006*

- Nishkam Ravi, Marco Gruteser and Liviu Iftode. Non-inference: An Information Flow Control Model for Location-based Services. *Proceedings of Third International Conference on Mobile and Ubiquitous Systems (Mobiquitous), San Jose, CA, July 2006*

- Michael Littman, Nishkam Ravi, Arjun Talwar and Martin Zinkevich. An Efficient Optimal-Equilibrium Algorithm for Two-Player Game Trees. *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence (UAI), Cambridge, MA, July 2006*

- Nishkam Ravi, Pravin Shankar, Andrew Frankel, Ahmed Algammal and Liviu Iftode. Indoor Localization Using Camera Phones. *Proceedings of the Seventh International Workshop on Mobile Computing, Systems and Applications (WMCSA/HotMobile), Semiahmoo, WA, April 2006*

- Nishkam Ravi, Marco Gruteser and Liviu Iftode. Information Flow Control for Location-based Services. *Poster in Mid-Atlantic Workshop on Programming Languages and Systems (MASPLAS), Piscataway, NJ, April 2006*

- Mandayam Raghunath, Nishkam Ravi, Chandra Narayanaswami and Marcel Rosu. Inverted Browser: A Novel Approach Towards Display Symbiosis. *Proceedings of the Fourth IEEE International Conference on Pervasive Computing and Communications (PerCom), Pisa, Italy, March 2006*

- Nishkam Ravi, Nikhil Dandekar, Preetham Mysore and Michael Littman. Activity Recognition from Accelerometer Data. *Proceedings of the Seventeenth Conference on Innovative Applications of Artificial Intelligence (IAAI/AAAI), Pittsburgh, PA, July 2005*

- Nishkam Ravi, Peter Stern, Nikel Desai and Liviu Iftode. Accessing Ubiquitous Services Using Smart Phones. *Proceedings of the Third International Conference on Pervasive Computing and Communications (PerCom), Kauai, Hawaii, March 2005*

- Nishkam Ravi, Cristian Borcea, Porlin Kang and Liviu Iftode. Portable Smart Messages for Ubiquitous Java-enabled Devices. *Proceedings of the First International Conference on Mobile and Ubiquitous Computing (Mobiquitous), Boston, MA, August 2004*

- Michael Littman, Nishkam Ravi, Eitan Fenson and Rich Howard. An Instance-based State Representation for Network Repair. *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI), San Jose, CA, July 2004*

- Michael Littman, Nishkam Ravi, Eitan Fenson and Rich Howard. Reinforcement Learning for Autonomic Network Repair. *Proceedings of the First International Conference on Autonomic Computing (ICAC) (Short paper), New York, NY, May 2004*

- Liviu Iftode, Cristian Borcea, Nishkam Ravi, Porlin Kang and Peng Zhou. Smart Phone: An Embedded System for Universal Interactions. *Proceedings of the Tenth International Workshop on Future Trends in Distributed Computing Systems (FTDCS), Suzhou, China, May 2004*