ABSTRACT

MURUKANNAIAH, PRADEEP KUMAR. Engineering Personal Agents: Toward Personalized, Context-Aware, and Privacy-Preserving Applications. (Under the direction of Dr. Munindar P. Singh.)

A personal agent acts and interacts on behalf of its human user. We envision a computing infrastructure in which autonomous personal agents realize social applications similar to how human societies operate. We develop computational tools, techniques, and abstractions to engineer personal agents. Personal agents enable smart spaces ranging homes, cars, and cities with applications spanning domains such as healthcare, entertainment, energy, and transportation.

Objectives: We seek to engineer personal agents that deliver a user- and context-specific experience, preserving their users' privacy. Engineering personal agents is challenging. On the one hand, a personal agent must capture its user's mental model of contextual needs—a challenge centered on human cognition. On the other hand, the agent must satisfy its user's needs in an automated manner, exploiting sensor data—a challenge centered on computational devices and infrastructure. Importantly, our approaches seek to shift the loci of computation from conceptually centralized servers of today to billions of user-controlled devices.

Methods: (1) We propose Xipho, an agent-oriented software engineering methodology for engineering personal agents. Xipho provides systematic steps to conceptualize a personal agent as a computational entity representing its user's goals, plans, contextual beliefs, and dependencies. By raising the level of abstraction, Xipho enables a developer to model and implement a personal agent close to how the developer naturally understands a user's requirements. (2) We describe Platys, a reusable middleware that provides architectural support for realizing personal agents. The Platys middleware efficiently gathers sensor readings about a user; maps low-level sensor readings to high-level abstractions of interest via active and semi-supervised machine learning; and exposes an API to simplify the development of personal agents. Platys is privacy-preserving as it runs locally on one or more personal devices of a user. (3) We describe

Muppet, a computational model that facilitates personal agents to preserve privacy in social settings, where information to be shared concerns multiple users. Muppet enables personal agents to compute an appropriate sharing policy in a multiuser scenario by exploiting the contexts, preferences, and arguments of the users involved in the scenario.

Findings: We demonstrate that our methods benefit both software developers and end-users via four human-subject studies—two developer studies, one end-user study, and one crowd study. All of our studies were approved by our university's Institutional Review Board (IRB).

Our first study, involving 46 developers, finds that Xipho (1) reduces time and effort required to model a personal agent, simplifying application development, and (2) yields agent designs easy for other developers to comprehend, simplifying application maintenance.

In our second study, an additional 46 developers employ the Platys middleware or Android framework to implement a personal agent. We find that Platys, compared to the Android baseline, (1) reduces development time and effort, (2) yields more modular agent implementations, and (3) enhances the usability and privacy-preserving aspects of a personal agent.

In our third study, 10 users employ Platys to collect sensor readings from smartphones and label contexts of their interests. From the data collected, we train Platys as well as two supervised and two unsupervised context recognition approaches. We find that Platys (1) recognizes contexts with higher accuracy than the unsupervised approaches compared with, and (2) requires fewer context labels to achieve a desired context recognition accuracy than the supervised approaches we considered, reducing the user effort required for training Platys.

Finally, to evaluate Muppet, we survey 988 Amazon Mechanical Turk users about a variety of multiuser scenarios and the optimal sharing policy for each scenario. Based on the participants' responses, we find that (1) contextual factors, user preferences, and arguments influence the optimal sharing policy in a multiuser scenario, and (2) employing features derived from the three types of information above, Muppet predicts the optimal sharing policy in a multiuser scenario with higher accuracy than baseline models.

Engineering Personal Agents: Toward Personalized, Context-Aware, and Privacy-Preserving Applications

by
Pradeep Kumar Murukannaiah

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina

2016

APPROVED BY:

Dr. Jon Doyle	Dr. James C. Lester
Dr. Timothy J. Menzies	Dr. Ranga Vatsavai

Dr. Munindar P. Singh
Chair of Advisory Committee

DEDICATION

To my grandfather Shree M. C. Lingappa.

ACKNOWLEDGEMENTS

I sincerely thank my advisor Professor Munindar P. Singh for his support and guidance. I am eternally indebted to him for all that I have learned from him.

I sincerely thank my doctoral committee members, Drs. Jon Doyle, James Lester, Tim Menzies, and Ranga Vatsavai. I immensely value their research feedback and career advice.

I am grateful to Drs. Tim Finin, Mike Huhns, Pankaj Mehra, Jose Such, Jessica Staddon, and Danny Weyns for providing me research and career advice on multiple occasions.

Special thanks to my wife Indumathi Srinivasachari and my sister Roopa Gopalaiah. I would not have achieved this feat without their support. I extend sincere gratitude to my mother-in-law and her family, and my sister's family for their immense support.

Over the years, I have worked with several colleagues at NC State including Qanita Ahmad, Nirav Ajmeri, Moin Ayazifar, Dr. Adrian Fernandez, Dr. Chung-Wei Hang, Dr. Anup Kalia, Ricard Lopez, Dr. Derek Sollenberger, Dr. Pankaj Telang, Dr. Guangchao Yuan, and Dr. Zhe Zhang. I have learned from each of them and sincerely appreciate their support.

Finally, thanks to the National Science Foundation (grant 0910868), NCSU Laboratory of Analytic Sciences, and NCSU Science of Security Lablet for providing financial support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Humans have an inherent understanding of the contexts in which they act and interact. A *personal agent* must adapt to the contexts of its human user. Our vision of personal agents enables a computational infrastructure in which the loci of computation are billions of user-controlled devices as opposed to a few conceptually centralized servers as it is today. Personal agents on our envisioned infrastructure support smart spaces (e.g., cars, homes, and cities) with applications spanning domains such as healthcare, transportation, and entertainment.

Our objective is to engineer personal agents such that they deliver a personalized, context-aware, and privacy-preserving user experience. Engineering a personal agents to do so is non-trivial. First, the personal agent must capture its user's mental model of contextual needs—a high-level concern centered on meaning. Second, the agent must acquire and satisfy user needs in an automated manner, exploiting sensor data—a low-level concern centered on devices and infrastructure. We seek to develop computational abstractions, methods, and techniques to address the challenges of engineering personal agents end to end.

We seek to answer four major questions relevant to the engineering of personal agents.

(1) What are the levels of abstraction (or granularity) at which a personal agent could employ the context information?

(2) How does the chosen level of context abstraction affect the quality of agents produced from the perspectives of both developers and end-users?

(3) How can a personal agent reason about the contexts of interest to its user at the desired level of abstraction from the sensor data?

(4) How can a personal agent preserve its users privacy as the agent (a) reasons about the user's personal data and (b) shares the user's personal information with other agents, e.g., to realize social applications?

We answer these questions via four contributions.

(1) We propose Xipho, an agent-oriented software engineering methodology for engineering personal agents. Xipho provides systematic steps to conceptualize a personal agent as a computational entity representing its user's goals, plans, contextual beliefs, and dependencies. By raising the level of abstraction, Xipho enables a developer to model and implement a personal agent close to how the developer naturally understands a user's requirements.

(2) We describe Platys, a reusable middleware that provides architectural support for realizing personal agents. Platys efficiently gathers sensor readings about a user; maps low-level sensor readings to high-level abstractions of interest via active and semi-supervised machine learning; and exposes an API to simplify the development of personal agents. Platys is privacy-preserving as it runs locally on one or more personal devices of a user.

(3) We describe Platys Reasoner, an active and semi-supervised machine learning technique to subjectively recognize a user's contexts. Our approach seeks to reduce the user effort required for training a model of contexts, and achieve better context prediction accuracy than traditional techniques.

(4) We describe Muppet, a computational model that facilitates personal agents to preserve privacy in social settings, where information to be shared concerns multiple users. Muppet

enables personal agents to compute an appropriate sharing policy in a multiuser scenario by exploiting the contexts, preferences, and arguments of the users involved in the scenario.

1.1 Xipho: A Methodology for Engineering Personal Agents

Humans have an inherent understanding of the contexts in which they act and interact. A personal agent adapts to the contexts of its human user. Engineering personal agents is nontrivial. First, a personal agent must capture its users' mental models of context—a high-level concern centered on meaning. Second, to be effective, the personal agent must acquire the desired contextual information automatically—a low-level concern centered on devices and infrastructure. We describe Xipho [87], a methodology for systematically developing a personal agent.

We treat context as a cognitive notion and understand other cognitive notions, such as goals and plans, as inherently related to context. Thus, a natural approach to developing a personal agent would be an agent-oriented software engineering (AOSE) methodology that employs cognitive notions throughout development. Existing AOSE methodologies describe generic steps of software development, but fall short in dealing with challenges specific to personal development. Xipho fills this gap by providing systematic steps for (a) capturing a personal agent's contextual requirements, (b) deriving a context information model specific to a personal, and (c) leveraging reusable components in a personal agent's design and implementation. These tasks pose nontrivial challenges and we demonstrate that Xipho addresses these challenges effectively.

A personal agent's contextual requirements are often (a) unknown ahead of time, and (b) subject to change as users employ the agent in different contexts. A developer, inevitably, makes assumptions about a personal agent's (and its user's) contextual needs, but often buries such assumptions in the implementation. Xipho provides constructs necessary to explicate such assumptions, yielding personal agents that are easy to maintain.

The space of what constitutes context is vast, e.g., a user's location, activities, emotions, or social setting. Existing techniques [9] model context as a notion generic across applications and users, which a developer must tailor to the requirements of a personal agent. Xipho assists a

developer in restricting the personal agent's context information model to a set of abstractions meaningful in the agent's scope, enabling the agent to offer a natural user experience.

To acquire the desired context information, a personal agent must address device and network centric concerns such as sensing, aggregating, and propagating context information. However, the concerns of context acquisition can be separated from those of personal agent development. Importantly, several existing works [6] address the problem of context acquisition. Xipho enables a developer to build a personal agent on reusable components, reducing the cognitive burden of agent development and yielding easily comprehensible agent designs.

Developer Study: We evaluated Xipho in a developer study comparing it with the baseline of Tropos [12], a well-known AOSE methodology (one factor with two alternatives study design). Our subjects were 46 students of a graduate level computer science course. We conducted the developer study in three phases: a *practice* phase in subjects learned and practiced Xipho or Tropos, a *modeling* phase in which subjects applied Xipho or Tropos to model a personal agent, and a *verification* phase in which subjects comprehended agents modeled by other developers.

Our findings suggest that Xipho, compared to Tropos, (a) reduces the time and effort required to develop a personal agent, and (b) yields agent designs that are easier for other developers to comprehend. We attribute these benefits to the systematic steps Xipho provides during early and late requirements engineering, architectural and detailed design, and implementation. Further, we conjecture that applications produced using Xipho offer a more natural user experience than those built conventionally.

1.2 Platys: A Middleware for Supporting Personal Agents

Personal agents are data driven, i.e., they make decisions based on the data their end-users produce. Although mobile and wearable devices are increasingly capable of sensing a variety of information about end-users' physiology, behavior, and environment, a user may not want to disclose such information to third-parties due to privacy concerns. Further, statistical analysis

of data aggregated from multiple users (as it is often done today) may not provide insights objectively valuable to all users. To address these challenges, we develop Platys middleware [88], which, (a) efficiently gathers data from multiple sensors into a user's personal data store shared across multiple user-controlled devices, (b) computes contextual information, e.g., places, activities, and social circles specific to the user, and (c) exposes the learned concepts to multiple personal agents, respecting the user's privacy policies.

The crux of Platys is to represent and compute context at a high-level abstraction. We demonstrate the value of a contextual high-level abstraction with respect one important aspect of a user's context: user *location*. In current practice, several software applications employ location at a low level of abstraction such as *position*, i.e., spatial coordinates (usually latitude and longitude). In contrast, Platys represents location as *place*, a high-level abstraction that represents a location not only by its spatial attributes, but also by users' actions and interactions associated with that location [86], e.g., *jogging* activity is an inherent attribute of a *jogging place*.

Platys describes a conceptual metamodel that unifies space, activities, and social circles into the notion of place. The three features of place are captured, not as independent entities, but in a unified manner.

Platys is extensible and captures places of interest to each user subjectively. Platys runs locally on a user's personal devices, promoting confidentiality and privacy. Further, each user can control which of his places an application can access and at what granularity.

Platys insulates application developers from dealing with low-level sensors and user-specific idiosyncrasies, and provides an end-user a single point of access to all of his personal data.

Developer Study: We evaluated Platys in a developer study in which subjects developed a locations-based application. The study was of one factor two alternatives design where subjects developed the prescribed location-based application by employing either (a) the low-level position abstraction provided by Android, or (b) Platys' high-level place abstraction.

Our study involved 46 students enrolled in a graduate-level computer science course. In addition to developing the application, subjects were asked to keep track of the time and effort they expended for development by answering a *time and effort survey* after each development session. We also analyzed the quality of the applications produced from the perspectives of both developers (via code metrics), and end-users (by studying the usability of the applications).

Our findings indicate that developers employing the Platys middleware spend significantly less time and effort than those not employing the middleware for representing and acquiring location, and enhancing the usability and privacy aspects of the application. Although developers employing the Platys middleware expend additional time and effort for acquiring the necessary background knowledge about the Platys middleware, the middleware pays off in other aspects of location-aware application development. Moreover, preparation is a one-time cost: a developer who employs Platys to develop several location-aware applications can save significant time and effort over the course of multiple applications. Our evaluation of the applications produced in the developer study indicate that location-aware applications produced using Platys are potentially (a) more usable and privacy-preserving from an end-user's perspective, and (b) easier to comprehend from a developer's perspective.

1.3 Platys Reasoner: Active Learning Context in Personal Agents

Platys Reasoner [88, 85, 136] is an important component of Platys middleware. The reasoner learns subjective context models from sensor data. Deriving useful and user-specific context models from a user's multimodal (originating from multiple sensors), sparse, and evolving sensor data is challenging. I developed Platys Reasoner, an *active* and *semi-supervised* machine learning approach, to map low-level sensor data to high-level contexts such as the user's places, activities, and social circles. Platys Reasoner offers distinct benefits over the traditional *supervised* (requiring user guidance) and *unsupervised* (not requiring user guidance) learning paradigms. Again, for concreteness, we focus on learning the place abstraction of context.

Traditional *unsupervised* place recognition techniques seek to learn patterns in the sensor

data, not requiring place labels. Typically, such approaches learn what we call as *staypoints*—sets of positions within a certain *radius* or those where a user stays for a certain *duration*. Although staypoint-based approaches do not require labeling, they have the following shortcomings. One, staypoints do not capture subjective nuances in how users perceive places since no fixed values for *radius* and *duration* yield desired places for all users. Two, a staypoint does not carry an inherent meaning. A user may eventually need a symbolic name (hence labeling) to distinguish staypoints. Three, staypoint-based approaches, often, require frequent sensor readings.However, sensing consumes battery power—a limited resource on mobile devices.

Alternatively, *supervised learning* techniques (e.g., classifiers) exploit user-provided place labels. Typically, training a classifier requires several training instances per class to produce good classification accuracy. However, acquiring training instances is challenging because place labeling requires user effort. Requiring each user to label each place of interest several times is not practically viable. Additionally, sensor readings are likely to be intermittent (all sensor readings may not be available when a user labels a place, e.g., GPS reception is limited indoors).

The Platys Reasoner seeks to address the shortcomings of the above approaches. Specifically, it combines two machine learning paradigms: (1) *active learning* [105] to reduce the labeling effort, and (2) *semi-supervised learning* [139] to efficiently deal with intermittent and infrequent sensor data.

The Platys Reasoner employs a classifier, albeit with additional steps in learning to address the challenges of traditional classification. Platys Reasoner's additional steps are motivated by the following intuitions. (1) Can we employ fewer training instances than traditionally required to train a classifier to achieve a desired accuracy? Yes, if we control what those training instances are (same number, though). (2) Can we exploit both labeled and unlabeled instances for training to achieve a better classification accuracy than training with labeled instances alone? Yes, if we exploit the hidden structure in the unlabeled data. (3) Can we exclude certain unlabeled sensor readings from training to enhance the classification accuracy? Yes, because many sensor readings may belong to none of the places a user labeled.

End-User Study: We evaluated Platys Reasoner via a user study. We analyzed the accuracy with which the reasoner recognizes places of interest to a user and its efficiency in doing so. No available datasets were adequate for our evaluation. We created our own dataset based on real traces collected from 10 users. Each user carried an Android phone installed with Platys middleware as his or her primary phone for three to 10 weeks. The middleware collected a user's place labels and recorded GPS, WiFi, and Bluetooth readings.

We treated place recognition as a classification problem and evaluated its performance via the metrics of *precision*, *recall*, and *F-measure*. We compared Platys' performance with that of two unsupervised and two supervised approaches. Our findings suggest that Platys Reasoner (a) reduces user effort required for training compared to two traditional supervised approaches, and (b) learns contexts with higher accuracy than two unsupervised approaches since it employs both unlabeled and labeled sensor data. Further, we found complimentary benefits of active and semi-supervised learning techniques. That is, whereas active learning was valuable in the initial phases of training, semi-supervised learning compensated for a user's non-compliance to place labeling requests in later phases of training.

1.4 Muppet: Engineering Personal Agents to Preserve Privacy

The Platys middleware promotes privacy by (a) running locally on user-controlled devices (not requiring users to share personal data with third parties), and (b) equipping users with fine grained control on contextual data (via a hierarchical model of context). In contrast to Platys, which deals with personal data, Muppet seeks to preserve privacy in a *multiuser* scenario, where information to be shared concerns multiple users. For example, suppose Alice uploads a photo from last weekend's party where she and her friend Bob appear together to a social network site, and tags Bob. Bob may find that the photo Alice uploaded is sensitive. However, since Bob has no control over uploading that photo, Alice's action threatens Bob's privacy.

Muppet is a decision-support system that helps users resolve multiuser privacy conflicts. The main challenge Muppet addresses is proposing an *optimal* solution: a solution most likely

to be accepted by all those involved in the multiuser scenario. Although an optimal solution may not exist for each multiuser scenario, identifying one, when it exists, can minimize the burden on the users to resolve the conflict manually.

Existing works [15, 51, 115] propose methods to automatically determine solutions to conflicts based on users' privacy preferences. These methods suffer from two main limitations. One, they always aggregate preferences in the same way regardless of the context or they only consider a very limited number of potential situations. Two, they do not consider the reasons behind users' preferences. However, evidence based on self-reported data [65, 131] suggests that users do listen to the explanations of others and that the optimal solution may depend on the particular context and reasons behind users' preferences. Following this idea, we empirically study three types of factors that potentially influence a privacy decision: the scenario's *context*, users' *preferences*, and their *arguments* [84] about those preferences. An argument is a justification a user employs to convince the others involved that the user's expectations are reasonable and should be taken into account for making a decision.

Our key objective is to build an argumentation-based model that accurately represents multiuser scenarios. To this end, we (1) identify important factors that potentially influence the inference; (2) evaluate the relative importance of these factors in inference; and (3) develop a computational model to predict an optimal solution in a given multiuser scenario.

Crowd Study: We design a study where human participants choose what they think is the most appropriate sharing policy in a multiuser scenario we specified. Combining different values of three types of factors (context, preferences, and arguments), we generate 2,160 scenarios. Considering the sheer number of participants required, we conducted our study on Amazon Mechanical Turk (MTurk), collecting responses from 988 unique MTurk participants.

Via a series of multinomial logistic regression models, we show that all three feature types we consider influence the optimal sharing policy. We find that (1) sensitivity has the highest influence on the optimal policy, among the contextual variables; (2) the most restrictive policy,

not the majority policy, has the highest influence on the optimal policy, among preference-based features; and finally, (3) users value arguments for sharing more than arguments for not sharing; however, if the argument for not sharing is an exceptional case users usually support not sharing.

By training an inference model on the data collected from MTurk participants, we find that a model employing argument-based features predicts optimal policy with higher accuracy than those not employing arguments. Further, from self reported data, we find that adding arguments increases a user's confidence in choosing the final sharing policy. Further, we investigate the data instances our prediction model misclassified to find out whether some combinations of scenario-defining elements make the prediction of optimal sharing policy difficult. We find that conflicts between arguments and preferences generate the majority of misclassification. This indicates that users interpret and prioritize arguments subjectively, which suggests that a system that aims at helping users solve multiuser conflicts has to be adaptive and learn from the user.

1.5 Summary

The variety and volume of personal data a user can produce is increasing with the emerging computing and communications infrastructure. Yet, traditional engineering approaches fail to yield applications fail to deliver user- and context-specific services. In contrast to the traditional approaches that rely on conceptually centralized servers, the methodology, middleware, and machine learning techniques we developed push the computation closer to users. In doing so, we facilitate personal agents catering to users' subjective and context-specific needs. Further, our techniques help preserve user privacy in both personal (Platys) and social (Muppet) settings.

Our work emphasizes rigorous empirical methods. We conduct four of user studies, demonstrating the benefits of our methods to both application developers (e.g., reduced development effort, ease of maintenance) and end-users (e.g., enhanced usability, reduced information overload). Our findings bear important implications for the practical adoption of our methods.

Chapter 2

Xipho: An Agent-Oriented Methodology

Humans have an inherent understanding of the contexts in which they act and interact. A *context-aware personal agent* (CPA) adapts to the contexts of its human user. We seek to engineer a CPA to (a) capture its users' mental models of context—a high-level concern centered on meaning, and (b) acquire the desired contextual information—a low-level concern centered on devices and infrastructure.

We describe Xipho, a methodology for systematically developing a CPA. Xipho treats context as a cognitive notion and understand other cognitive notions, such as goals and plans, as inherently related to context. Thus, a natural approach to developing a CPA would be an agent-oriented software engineering (AOSE) methodology that employs cognitive notions throughout development. Existing AOSE methodologies, e.g., [12, 94, 129], describe generic steps of software development, but fall short in dealing with challenges specific to CPA development. Xipho fills this gap by providing systematic steps for (a) capturing a CPA's contextual requirements, (b) deriving a context information model specific to a CPA, and (c) leveraging reusable components in a CPA's design and implementation. These tasks pose nontrivial challenges (described below). We demonstrate that Xipho addresses the challenges of CPA development successfully.

A CPA's contextual requirements are often (a) unknown ahead of time, and (b) subject to change as users employ the CPA in different contexts. A developer, inevitably, makes assumptions about a CPA's users' contextual needs, but often buries such assumptions in the implementation. Xipho provides constructs necessary to explicate such assumptions, yielding easily renewable CPAs [108].

Context is traditionally defined as "any information relevant to an interaction between a user and an application" [20]. Accordingly, the space of what constitutes context is vast, e.g., a user's location, activities, emotions, or social setting. Existing techniques [9] model context as a notion generic across applications and users, which a developer must tailor to the requirements of a CPA. Xipho assists a developer in restricting a CPA's context information model to a set of abstractions meaningful in the CPA's scope, enabling the CPA to offer a natural user experience.

To acquire the desired context information, a CPA must address device and network centric concerns such as sensing, aggregating, and propagating context information. However, the concerns of context acquisition can be separated from those of CPA development. Importantly, several existing works address the problem of context acquisition [6]. Xipho enables a developer to build a CPA on reusable components, reducing the cognitive burden of CPA development and yielding easily comprehensible CPA designs.

Contributions: Our contributions are two fold. First, we propose Xipho as an extension of Tropos for developing CPAs. Xipho addresses nontrivial challenges associated with requirements acquisition, design, and implementation of a CPA. We demonstrate Xipho via a case study, which that Xipho (a) reduces the time and effort required to develop a CPA, and (b) yields CPA designs that are easy for other developers to comprehend.

2.1 Background: Tropos

Xipho extends Tropos [12] to support CPA development. Specifically, Xipho adopts the Tropos metamodel, which consists of the following main constructs.

Actor: A social, physical, or software agent (or a *role* of an agent). An actor has goals within a system.

Goal: A strategic interest of an actor. A *hard goal* has a crisp satisfactory condition. A *soft goal* has no clear-cut definition or criterion for deciding whether it is satisfied or not. Thus hard goals are *satisfied* whereas soft goals are *satisficed*.

Plan: An abstraction of doing something. Executing a plan is a means of satisfying or satisficing a goal.

Resource: A physical or information entity.

Dependency: A relationship indicating that a *depender* actor depends on a *dependee* actor to accomplish a goal, execute a plan, or furnish a resource. The object (goal, plan, or resource) here is the *dependum*.

Belief: An actor's representation of the world.

Capability: An actor's ability to choose and execute a plan to fulfill a goal, given certain beliefs and resources.

Our motivation to extend Tropos is that it spans all development phases. First, Tropos captures *early* and *late* requirements as *system-as-is* and *system-to-be* models, respectively. A system-as-is model captures (a) the actors involved: primary users of the application as well as those indirectly affected by it, (b) goals and plans of each actor, and (c) dependencies among actors on their goals or plans. A system-to-be model introduces the solution as the system-to-be

actor and its goals, plans, and dependencies. In the following phases, Tropos maps the system-to-be actor into one or more agents, and derives a detailed specification of agents' capabilities to implement on a chosen platform.

2.2 Xipho

Xipho augments Tropos with tasks specific to CPA development in each development phase as shown in Table 2.1. Before describing Xipho's steps, we introduce a case study, and derive its system-as-is and system-to-be models.

Table 2.1: An overview of how Xipho extends Tropos to support CPA development.

Development Phase	**Tropos Task**	**Xipho Task (Extension)**
Early & late requirements	Identify system-as-is and system-to-be actors, and their goals, plans, and dependencies	Step 1: Identify contextual beliefs and resources
Architectural design	Define system's global architecture	Step 2: Derive a context information model
	Map system actors to software agents	Step 3: Incorporate a middleware agent
Detailed design	Specify agent capabilities and interactions	Step 4: Specify contextual capabilities
Implementation	Implement agents on a chosen platform	Step 5: Implement contextual capabilities

Case Study: Ringer Manager

We demonstrate Xipho via a case study, which involves engineering the *Ringer Manager Application (RMA)*, a CPA for phones to help a user better handle incoming calls.

Automated ringer mode. The RMA sets an appropriate ringer mode on the user's phone based on the user's context at the time of an incoming call. The alternatives (assumed to be mutually exclusive) are to set the ringer mode to *silent*, *vibrate*, or *loud*.

Automated notification. If the user does not answer the call, the RMA sends a notification to the caller. The notification can be generic (e.g., "leave your name and number"), or describe the user's context at the time of missing the call, either abstractly (e.g., "in a meeting") or in detail (e.g., "in a meeting with Bob at the Starbucks").

We imagine the system-as-is as follows. Each episode starts when a *caller* tries to reach a *callee* by phone. The callee wants to be reachable unless he wants to work uninterrupted. The callee's plan is to answer the call if he wants to be reachable and not answer otherwise. The callee's decision to answer or not answer depends on (a) whether he disturbs a *neighbor* by answering or (b) if the caller has a pressing need to reach him. In these cases, the callee depends on the neighbor or the caller to provide appropriate information. The callee sets an appropriate ringer mode on his phone to help him answer or not (e.g., loud to answer; silent to not). Next, when the callee does not answer a call, he would notify the caller of a reason (e.g., busy, in a class, talking to boss, and so on) or to ignore the call depending on who the caller is. If he decides to notify the caller, he would disclose only the necessary details, preserving privacy. Figure 2.1 shows the system-as-is model.

However, the system-as-is is quite inefficient. First, it relies on the callee to manually set an appropriate ringer mode and send an appropriate notification. Second, a caller has no effective way of expressing an urgent need to reach the callee—calling repeatedly does not help if the phone is silent. Third, it is tedious for a neighbor to let each callee know that the neighbor prefers not to be disturbed.

Figure 2.2 introduces the RMA (system-to-be) actor. The RMA acts on behalf of the callee and accordingly adopts the callee's goals to make call handling more efficient. This model captures what the RMA does and why, by linking the RMA's plans to the callee's goals (e.g., *set as silent* to accomplish the callee's goal *to be uninterrupted*). For simplicity, we explore the RMA only from the callee's (primary user's) perspective and assume that the secondary users (caller and neighbor) employ appropriate mechanisms to provide the information required by the RMA.

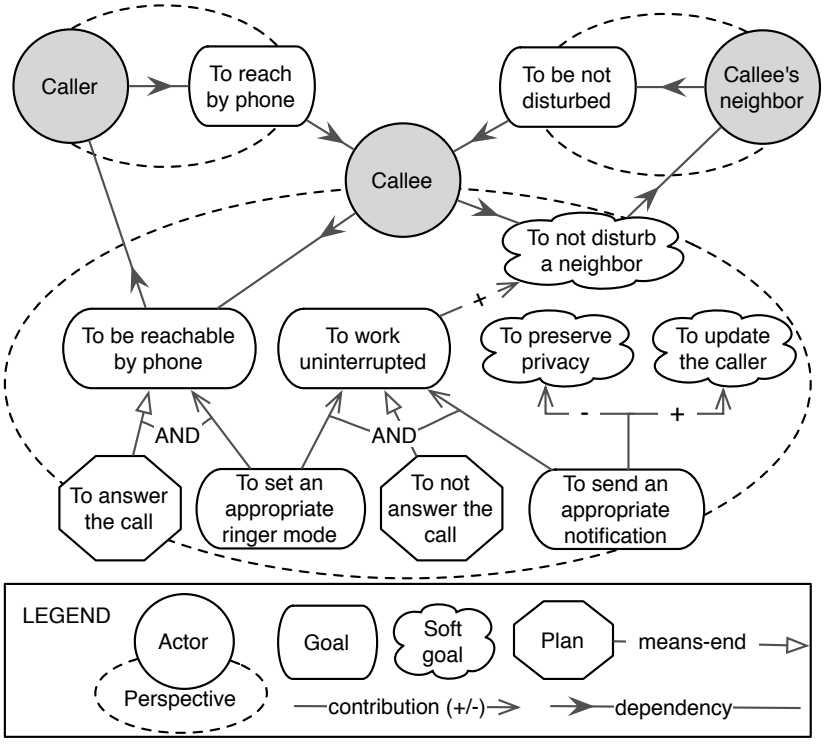


Figure 2.1: Actor model: Callee's perspective.

2.2.1 Step 1: Context-Means Analysis

A CPA needs to make decisions based on the contexts of one or more of the actors involved. The objective of *context-means analysis* is to identify scenarios where an actor's context provides the application a means of making a decision. We describe four types of such scenarios with examples.

OR-decomposition, where some decomposed goals or plans are to be chosen over the rest based on context, e.g., whether to set the ringer mode as *silent* or *vibrate*—vibrate may be acceptable in a seminar, but not in a one-on-one meeting.

Conflicting goals, where the goal chosen for accomplishment depends on context. For example, the RMA must choose between the callee's goals *to be reachable* and *to work uninterrupted*. This decision could be based on the callee's context—busy or not.

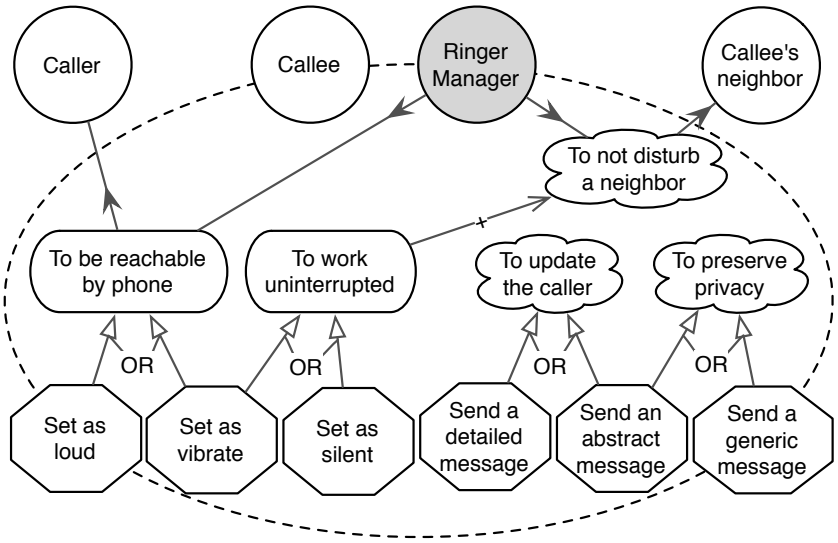


Figure 2.2: Expanding the RMA's perspective.

Soft goal, where the extent to which the goal is satisficed depends on context. For example, each ringer mode satisfices the soft goal *to not disturb a neighbor* differently. The chosen ringer mode could be based on who the neighbor is, e.g., set as *silent* near a colleague, *vibrate* near a family member, and *loud* near a stranger.

Dependency, where the dependum can be refined based on context. For example, the dependum *to be reachable* can be refined to context because the actual dependency is that the callee depends on the caller to provide contextual information (e.g., caller's context indicating whether he has a pressing need to reach the callee).

A developer must identify all scenarios of each type described above and perform one of the following for each.

- If the scenario is influenced by the primary user's (*callee*) context, capture the influence as a belief and add context-means links from the belief to each goal or plan involved in the scenario.
- If the scenario is influenced by a secondary user's (*caller* or *neighbor*) context, capture

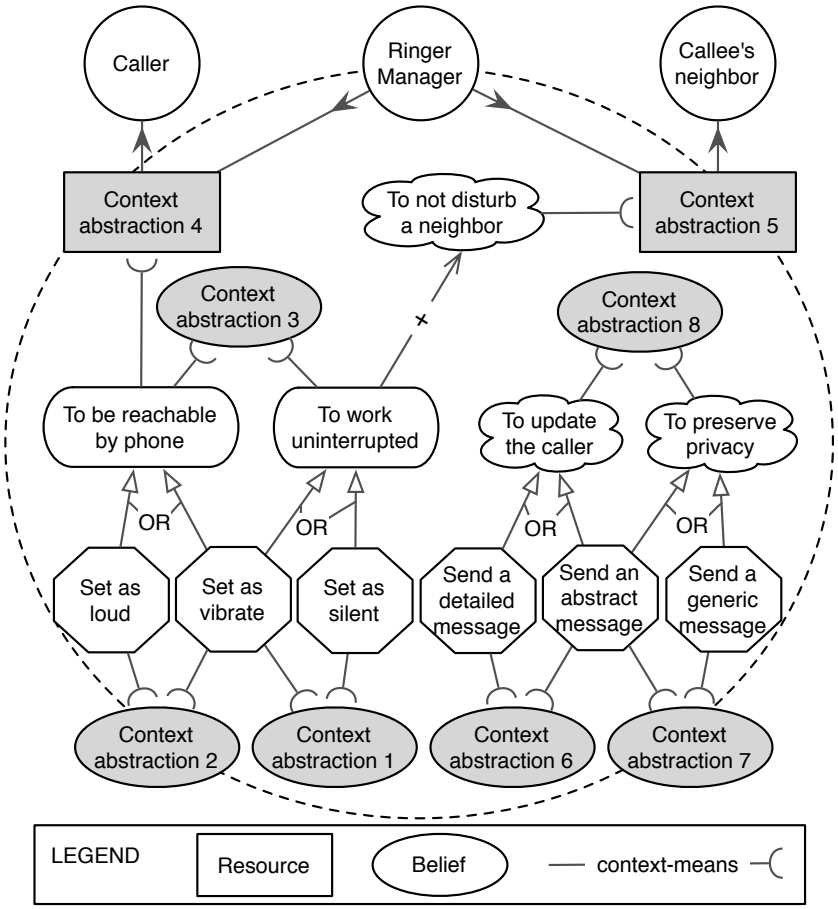


Figure 2.3: The RMA after context-means analysis.

the context as a resource and add (or update) a dependency with the context resource as the dependum. Also, add context-means links from the resource to each goal or plan involved in the scenario.

The motivation is that the CPA would maintain beliefs about its primary user, and access information resources about the secondary users as shown in Figure 2.3.

2.2.2 Step 2: Context Information Modeling

We used a *context abstraction* as a placeholder for each contextual belief or resource in the previous step. Now, we systematically refine these abstractions in the CPA's scope for two

reasons: (a) a motivation for model-driven development is to document the developer's thought process by explicitly capturing his or her intuitions, and (b) a detailed model would yield a detailed specification, potentially making the implementation easier. To derive a CPA's context information model, identify:

A context model for each generic abstraction found in the previous step such that the context model consists of cognitive abstractions sufficient to make the decision specific to the context-based scenario influenced by the corresponding generic abstraction. For example, in Figure 2.3, *Context abstraction 2* is used to decide whether to set the ringer mode as *loud* or *vibrate*. This generic abstraction can be modeled as a spatial abstraction, *Ambience*, assuming that all that matters in setting the ringer mode as *loud* or *vibrate* is the callee's ambience. The decision to not set as *silent* must already have been made, using *Context abstraction 3*.

Context instances of a context abstraction, where appropriate, such that a context instance represents all situations, within the scope of the corresponding abstraction, that lead to the same context-based decision. For example, context instances of *Ambience* could be *Noisy* and *Quiet* assuming that the ringer mode would be *vibrate* if the ambience is *Quiet* and *loud* if *Noisy*. However, to gain adaptivity, it would be desirable to elicit the instances of an abstraction at run time. We refer to such an abstraction as an *open abstraction*. For example, modeling the callee's *activity* as an open abstraction helps us elicit from the user which of his activities are to be uninterrupted.

Figure 2.4 shows a context information model derived for the RMA (each context instance is highlighted with a thick border). We restrict the case study to the ringer management aspect of the RMA (omitting notifications) for simplicity. We employed the context abstractions of *space*, *activity*, and *social circle* described in an existing metamodel [88], which however is loosely coupled with Xipho.

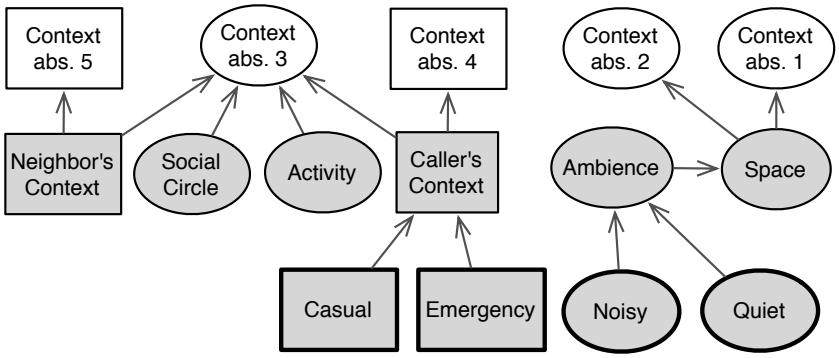


Figure 2.4: RMA's context information model.

2.2.3 Step 3: Context Middleware

Once we derive a CPA-specific context information model, how can we enable the CPA to acquire this information? To acquire context information, the CPA must:

Elicit context instances of each open context abstraction from the primary user, e.g., elicit from the callee that the desired instances of the *Social circle* abstraction are *Family*, *Colleagues*, and *Others*.

Recognize context instances of each context abstraction from sensors, e.g., recognize the callee's *Ambience* as *Quiet* via data from the microphone on the callee's phone.

Acquire context resources from the secondary users, e.g., acquire from the caller that the *Caller's context* is *Casual*.

These are nontrivial tasks, but common to all CPAs. Although we model context information specific to each CPA, the abstractions can overlap as a user employs multiple CPAs (especially, since the abstractions are high-level, cognitive constructs). Thus, Xipho incorporates a middleware architecture in which (a) a reusable middleware agent elicits and recognizes a user's contexts, and (b) multiple CPAs interact with the middleware to acquire desired contexts.

Figure 2.5 shows the middleware actor in the RMA's architecture. Importantly, such middleware components have been realized, e.g., [6, 72, 88]. Xipho can incorporate any middleware

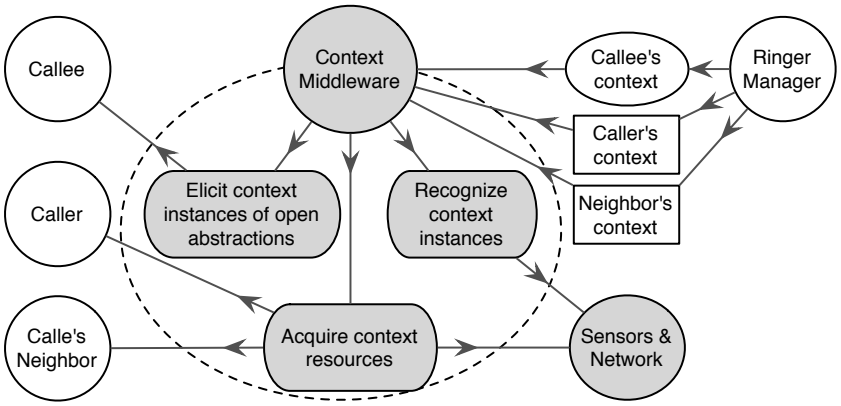


Figure 2.5: Introducing a middleware actor.

that achieves the goals specified in Figure 2.5.

2.2.4 Step 4: Contextual Capability Modeling

The objective of this step is to obtain a detailed agent specification to simplify the implementation. Xipho's specification of a CPA is a set of *contextual capabilities*, each conditioned on a context abstraction at an instance. A contextual capability can be represented as a rule, e.g., *Ambience = Noisy* $\rightarrow$ *Set as loud*. A developer performs the following substeps to derive an application specification.

- Identify agent capabilities, e.g., choosing and executing a plan, choosing a goal to accomplish, and managing a dependency.
- Identify the context abstraction that conditions each capability, replacing the generic context abstraction placeholders of Step 1 with specific abstractions from Step 2.

Figure 2.6 shows a refined actor model of the RMA that combines context abstractions and (highlighted) capabilities. This model can be used to generate a detailed set of contextual capabilities. Table 2.2 lists the contextual capabilities we identified for the RMA. We use variables ($?A_1$, $?S_1$, etc.) to capture instances of open abstractions.

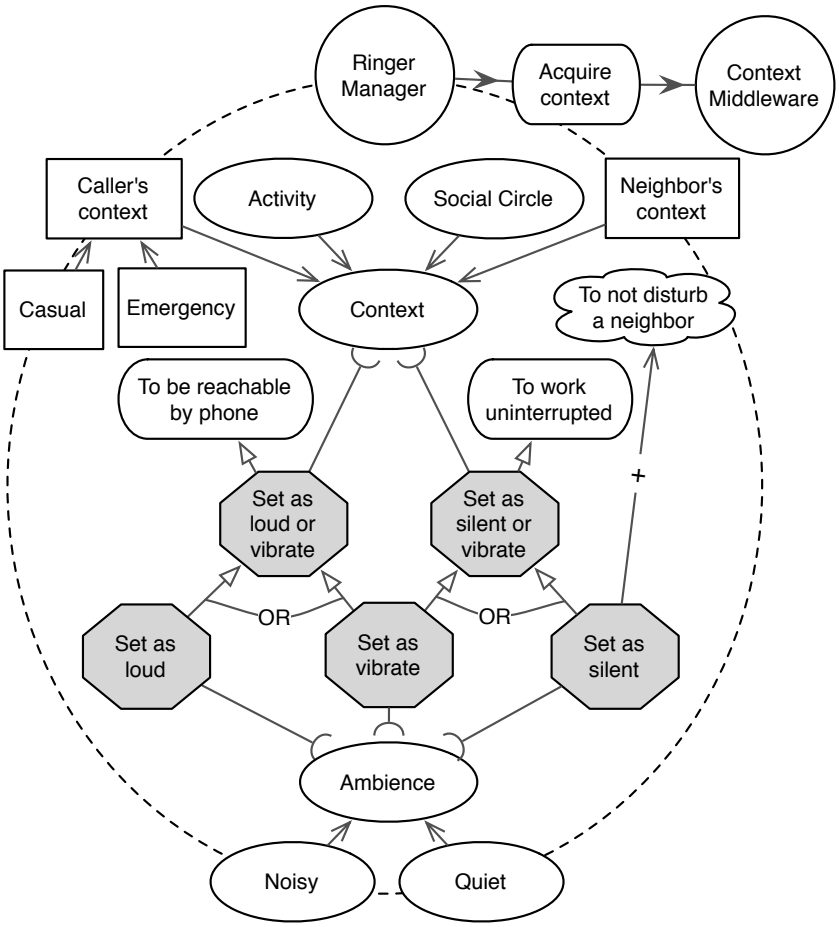


Figure 2.6: RMA's final model.

2.2.5 Step 5: Implementation

Now, a developer must implement techniques to

Interact with the middleware to determine (a) elicited instances of open abstractions, e.g., to determine that instances of *Activity* are *Working*, *Driving*, and *Dining*, and (b) the instance at which an abstraction is at a given time, e.g., to determine if $Activity = Working$, now.

Substitute variables in a contextual capability with instances elicited by the middleware, e.g., substitute $Activity = ?A_1$ with $Activity = Driving \vee Activity = Dining$.

Table 2.2: RMA specification.

ID	Contextual Condition	→ Capability
C_1	*Activity* = $?A_1$ ∧ *Social circle* = $?S_1$ ∧ *Neighbor's context* = $?N_1$ ∧ *Caller's context* = *Emergency*	→ *Set as loud* ∨ *Set as vibrate*
C_2	*Activity* = $?A_2$ ∧ *Social circle* = $?S_2$ ∧ *Neighbor's context* = $?N_2$ ∧ *Caller's context* = *Casual*	→ *Set as silent* ∨ *Set as vibrate*
C_3	C_1 ∧ *Ambience* = *Quiet*	→ *Set as vibrate*
C_4	C_1 ∧ *Ambience* = *Noisy*	→ *Set as loud*
C_5	C_2 ∧ *Ambience* = *Quiet*	→ *Set as silent*
C_6	C_2 ∧ *Ambience* = *Noisy*	→ *Set as vibrate*

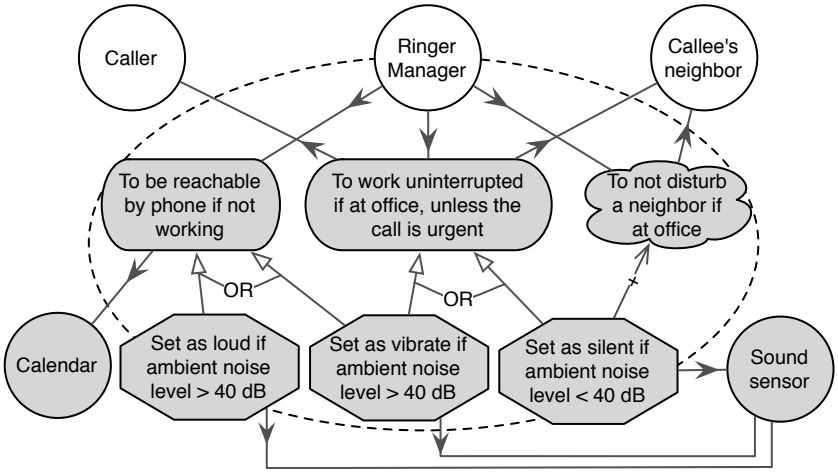


Figure 2.7: A model derived without Xipho.

Exercise capabilities by executing the rules derived in the previous section, e.g., exercise *Activity* = *Working* → *Set as silent* ∨ *Set as vibrate*.

2.3 Comparison

Xipho is an AOSE methodology tailored specifically for CPA development. However, what are the benefits of Xipho? To answer this, we compare the Xipho's RMA model shown in Figure 2.6 with an alternative model shown in Figure 2.7.

The alternative model enhances goals and plans in Figure 2.2 to incorporate context. The

alternative model represents three major mistakes a developer, not applying Xipho, can commit. The alternative model:

Lacks explicit semantics of context, whereas Xipho models convey the semantics described in Steps 1 and 2. Such lack leads to ambiguity in interpreting a model. For example, Figure 2.7 is not clear about whether two goals (e.g., *...if not working...* and *...if at office...*) conflict; similarly, the figure does not convey that the ringer mode must be *silent* only if the call is not urgent and the callee's ambience is quiet.

Introduces redundancy in the context information model, whereas Xipho removes it by incorporating cognitive constructs. For example, each goal in Figure 2.7 describes a contextual condition involving activities, redundantly. In contrast, Figure 2.6 simplifies the model by including the *Activity* open abstraction.

Does not separate the concerns of context acquisition from the RMA design, whereas Xipho cleanly separates the RMA's and middleware's concerns. For example, Figure 2.7 describes noise levels used to decide a ringer mode within RMA's design. In contrast, Figure 2.6 delegates such tasks to the middleware, simplifying the RMA's design.

2.4 Empirical Evaluation

The foregoing intuitions lead us to hypothesize that Xipho (a) reduces the time and effort required to develop a CPA, and (b) enhances the comprehensibility of CPA designs (for other developers). We evaluated our hypotheses in a developer study, comparing Xipho against the baseline of Tropos.

2.4.1 Study Design

Our subjects were 46 students of a graduate level computer science course. Each subject worked in a team of three (12 teams), two (three teams), or one (four subjects) of his or her choosing.

Subjects received a partial grade toward course credit for producing all deliverables. The study was approved by the Institutional Review Board (IRB) and we obtained an informed consent from each subject. Nonparticipants could work on an alternative task.

We conducted the developer study in three phases.

Practice, which required each team to learn one of Tropos or Xipho, and exercise it to model an application, which we reviewed to help them understand.

Modeling, which required each team to model a second application. No feedback was given during this phase.

Verification, which required each subject (working solo) to verify an existing model of a third application for completeness and comprehensibility.

The study lasted for eight weeks—two weeks per phase for subjects and a week each between phases for us to review.

Study Units

Subjects completed a prestudy survey about their experience in software modeling and development (of context-aware and mobile applications, and in general). Using this information, we formed three groups of teams such that teams' skill sets and sizes balanced across groups.

In the first phase, we assigned one of the following applications to each group. In the subsequent phases, we rotated the assignments as shown in Table 2.3. For each application, we provided the following description and three–four scenarios in which the application must employ the (primary or secondary) user's context.

Smart alarm, whose objective is to advance or postpone an alarm on a user's phone. The application must employ context information such as (a) traffic and weather conditions, (b) the amount of rest the user has had by the time alarm goes off, and (c) changes to the user's schedule of upcoming events.

Lifestyle motivator, which helps a user find friends and identify physical activities to perform with those friends. The application must employ context information such as (a) the proximity to appropriate facilities and desired friends, (b) the weather conditions, and (c) the amount of exercise the user has already had.

Intelligent reminder, which decides when and how to remind a user of events. The application must employ context information such as (a) convenient resources (e.g., remind on the computer instead of the phone), (b) importance (e.g., reply to the boss), and (c) timeliness (e.g., pick up a book when near the library).

Table 2.3: Application assignments in each phase.

Phase	Task	Group 1	Group 2	Group 3
1	Practice	Alarm	Motivator	Reminder
2	Modeling	Motivator	Reminder	Alarm
3	Verification	Reminder	Alarm	Motivator

Alternatives

Teams within each group (1, 2, or 3) were divided into two groups (again, with balanced skill sets and team sizes).

Control group (C): provided with the modeling primitives, a description, and example models from Tropos.

Xipho group (X): provided with the steps, modeling primitives, and the RMA example.

Deliverables

In each of the first and second phases, subjects were given an application specification (and a methodology) and asked to deliver (a) a model of the application (covering all specified

scenarios); (b) responses to a survey after each work-session recording the time spent (in hours and minutes), and effort expended (on a scale of 1∼*easy* to 7∼*difficult*) during the session. The survey also required subjects to record a detailed breakdown of tasks they performed in each session.

In the third phase, each subject was given the specification of a third application and two models of it produced by other subjects (one from Control and one from Xipho team) in the second phase. A description of the Xipho's primitives was provided to those who did not apply Xipho in earlier phases. Our motivation was to compare the comprehensibility of models by subjects experienced with Xipho and those new to it. The subjects were asked to verify the model and answer a survey to record (a) a rating of how complete the model was (on a scale of 1∼*incomplete* to 7∼*complete*), and (b) a rating of how comprehensible a model was (on a scale of 1∼*easy* to 7∼*difficult*) relative to each context-based application scenario the model handled.

We allowed teamwork in the first two phases, but required solo work in the third because (a) such ratings are inherently subjective, and (b) the third phase was less demanding than the first two, and we expected subjects to complete the deliverables (in time) without compromising quality.

2.4.2 Results

We performed two-tailed (a) t-test to compare the difference in mean (μ) time spent, (b) F-test to compare the difference in the variances (σ^2) of time spent, and (c) Wilcoxon's $ranksum$-test to compare the difference in the median ($\widetilde{x}$) of effort ratings during Practice and Modeling [55]. We compared medians for the effort ratings since rating is ordinal.

With respect to the Practice and Modeling phases, Figure 2.8 shows box plots of times spent, and Figure 2.9 shows effort perceived, building models (there were no instances of the lowest rating). The figures also show the result of hypothesis testing (* and ** indicate sufficient evidence to reject the corresponding null hypothesis at significance levels of 10% and 5%,

respectively).

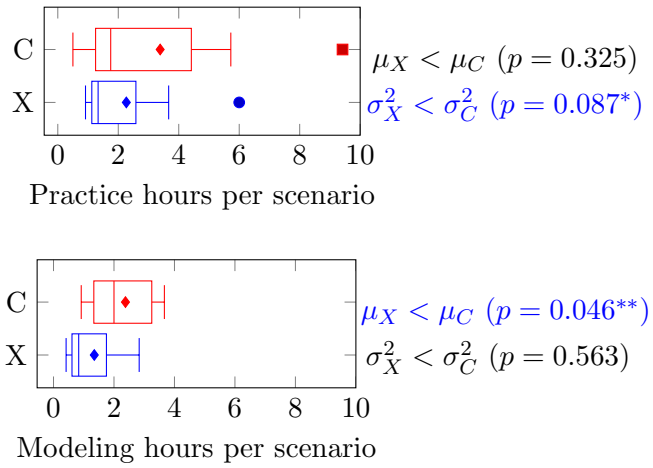


Figure 2.8: Time spent building models.

Recall that each model produced in Modeling was rated by more than one subject for completeness and comprehensibility during Verification. First, we measured the interrater reliability for ratings of each model using Krippendorff's alph (α) [60] ($\alpha = 1$ indicates perfect reliability and $\alpha = 0$, the absence of reliability; $\alpha = 0.67$ is a suggested lower bound for filtering). We excluded all models with $\alpha < 0.67$ from further analysis. Next, we performed *ranksum*-tests to test if the ratings differed significantly. Figure 2.10 shows a comparison, and the results of hypothesis testing.

2.4.3 Discussion

Time and Effort of Model Building

We found that the time and effort expended by a Xipho team in the Modeling phase are significantly lower than those expended by a Control team ($p = 0.046$ and $p = 0.01$, respectively). This validates that Xipho, a methodology tailored for CPA development, can simplify the development process as opposed to a generic methodology.

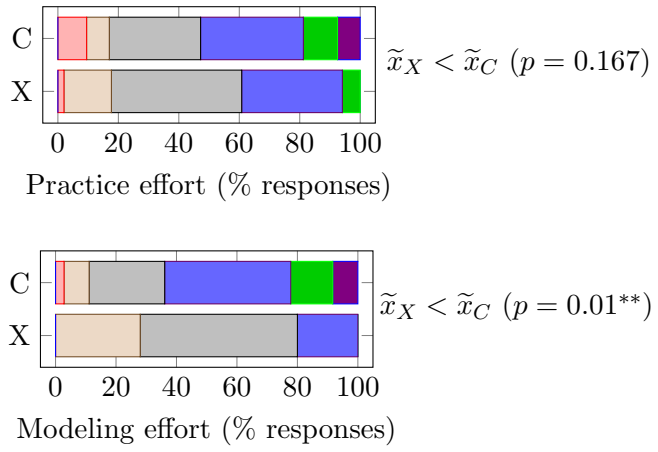


Figure 2.9: Perceived effort in modeling.

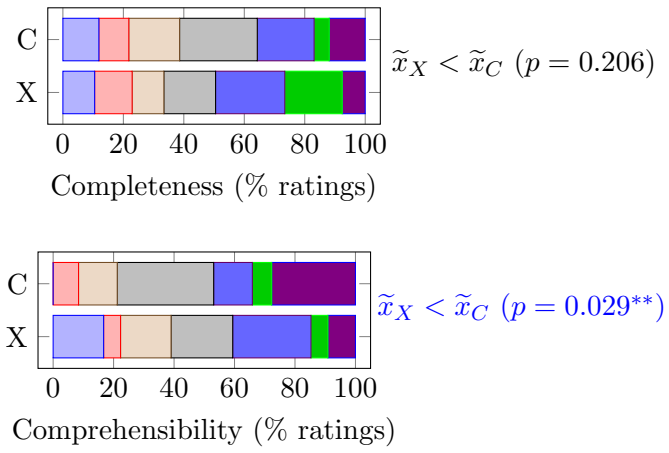


Figure 2.10: Subjective ratings for completeness and comprehensibility.

Next, we found that the differences in time spent and effort were not significantly different during the Practice phase (when teams were learning Xipho or Tropos). However, we made an important observation—the variance in time spent was significantly larger for Control than Xipho teams, although Control teams had fewer primitives to learn than Xipho teams. We speculate that such difficulties could lead a developer, especially a beginner, to give up modeling. Thus, a methodology with systematic steps specific to CPAs is essential for the success of model-driven development of CPAs.

Completeness and Comprehensibility

We found that the completeness ratings of Xipho and Control models did not differ significantly. The result was not surprising, though, for two reasons. (1) Conceptually, the modeling primitives of Tropos are sufficient to model any of the assigned applications. Thus, neither of the groups has an advantage from this perspective. (2) Each team, irrespective of the group, had an incentive (project grade) to produce complete models (even if it took more time).

Next, we found that comprehending a Xipho model could be significantly easier than a Control model. This is an important result since Xipho's ability to yield comprehensible models is a major benefit of employing it. To understand why this is the case, we compared the modeling constructs used by Control and Xipho teams, as shown in Table 2.4. We found that compared to Control, Xipho models employed:

Shorter textual descriptions within primitives. We attribute a Xipho model's conciseness to employing the explicit context modeling semantics provided by Xipho. Further, our results suggest that concise Xipho models are easier to comprehend than Control models' subjective and verbose textual descriptions.

Fewer modeling primitives indicating that Xipho models reused contextual beliefs and resources across goals and plans. We conjecture that employing fewer primitives helps reduce information overload and leads to models that are easier to comprehend.

Fewer actors and dependencies indicating that Xipho models delegated the concerns of context acquisition to the middleware, whereas Control models employed additional actors (with associated, goals and plans) to acquire context. Also, a Control developer must deal with context acquisition for each application, whereas a Xipho developer needs to understand the middleware just once.

Table 2.4: Modeling primitives employed.

Primitive	**Count**		*p*-value	**Text len.**		*p*-value
	C	**X**	$\tilde{\mathbf{x}}_{\mathbf{X}} < \tilde{\mathbf{x}}_{\mathbf{C}}$	**C**	**X**	$\tilde{\mathbf{x}}_{\mathbf{X}} < \tilde{\mathbf{x}}_{\mathbf{C}}$
Actor	6.0	3.4	0.02**	77	33	0.02**
Dependency	8.2	4.2	0.02**	–	–	–
Goal	4.4	2.6	0.57	136	58	0.13
Soft goal	5.8	0.8	0.01**	231	21	0.01**
Plan	9.8	7.4	0.29	365	145	0.06*
Belief	0.0	3.6	–	0	45	–
Resource	0.0	2.0	–	0	30	–
Overall	*34.2*	*24*	*0.09**	*809*	*332*	*0.02***

2.5 Related Work

Xipho begins with cognitive notions and systematically leads to an opportunity to exploit reusable components. Xipho helps bridge AOSE and context-aware systems.

Context-Aware Systems

A wealth of context-aware systems research [6] focuses on providing architectural support for developing context-aware applications, usually via a middleware, e.g., [49, 72, 88, 106], to protect developers from low-level concerns of context acquisition. Often, such architectures provide a generic context metamodel [9], which an application developer must tailor during development. Sollenberger and Singh [111] show that developers employing a systematic methodology benefit

more from a middleware than those who do not. CPA developers can employ Xipho with any middleware that can elicit and reason about context instances. Bolchini et al. [11] describe a context-aware system that centrally manages contextual data. In contrast, Xipho's middleware agent manages a user's contextual data locally and advocates users to interact for satisfying dependencies on contextual resources.

AOSE Methodologies

AOSE promotes development activities to capture high-level abstractions, e.g., agents and goals over low-level abstractions, e.g., classes and methods. Xipho extends Tropos to explicitly deal with the cognitive notion of context.

Ali et al. [2] capture context-based scenarios as variation points and employ them to analyze requirements. Whereas their focus is to analyze a contextual goal model to detect conflicts and inconsistencies among requirements, Xipho deals with context in requirements as well as in design and implementation. Zacarias et al. [135] describe a context-aware agent-oriented ontology for modeling human agents in an enterprise setting. Whereas they provide a metamodel (ontology) to describe the dynamic and situated human behavior, Xipho provides systematic steps to exploit such a metamodel for developing CPAs.

Xipho's context analysis, modeling, and specification steps are generic, and can be employed in other AOSE methodologies. Prometheus [129] can be extended to incorporate context as percepts and external data used to describe the environment. Xipho can be incorporated into ADELFE [7] to characterize the environment, where ADELFE's active and passive entities correspond to Xipho's context instances and open abstractions. Rahwan et al. [94] describe a social modeling extension to the ROADMAP methodology. Xipho brings together a social model, users' activities, and spatial attributes into a context model, which can be linked a role's responsibilities, generalizing Rahwan et al.'s extension.

Other Methodologies

Ceri et al. [16] employ WebML to model context and context-triggered adaptive actions in a user-independent manner. Serral et al. [104] employ a domain-specific PervML to model context followed by auto generation of code that represents and handles context. Henricksen and Indulska [46] employ CML (Context Modeling Language) to model context and preference modeling to rank context-based choices of a user. These methodologies describe how to model context, assuming that a developer knows what aspects of context to model and why. Xipho helps a developer not only to model context, but also to arrive at meaningful context abstractions and instances specific to an application scenario.

2.6 Conclusions and Directions

This paper has shown how an agent-oriented methodology can be enhanced to engineer CPAs, an important problem that has not been adequately addressed in the literature. Our findings about the benefits of Xipho bear important implications for its practical adoption. A huge market for CPAs already exists in the form of smart phone applications (such as RMA). Further, we conjecture that applications produced using Xipho offer a more natural user experience than those built conventionally. We are considering ways to design a tractable study to evaluate this claim.

Xipho provides a key component of our overall vision of developing CPAs. On the one hand, Xipho extends the benefits of a goal-based methodology (specifically, Tropos) to CPAs—context is arguably naturally understood via cognitive notions. On the other hand, Xipho builds a CPA on a reusable middleware, separating the concerns of context acquisition from CPA development. Our implementation of the middleware elicits a user's context instances and learns to recognize them from sporadic sensor data via semi-supervised machine learning [39]. Our ongoing work seeks to reduce user effort in context elicitation via active learning.

Systematically contextualizing cognitive notions such as privacy and trust is largely unex-

plored. A contextual approach to privacy and trust could facilitate applications with multiple interacting CPAs. Such an extension could build on existing approaches for finding social relationships between agents sharing a context [85], and coordinating agents by transparently propagating the context information [72].

Acknowledgments

Thanks to Raian Ali, Fatma Başak Aydemir, John Mylopoulos, and the anonymous reviewers for helpful comments.

Chapter 3

Platys: A Middleware Architecture

In current practice, several software applications employ contexts at a low level of abstraction such as *position*, i.e., spatial coordinates (usually latitude and longitude). We imagine that position is popular because it matches existing location acquisition techniques including Global Positioning System (GPS), and cellular and WiFi triangulation [62]. Current mobile devices provide hardware (built-in sensors) and software (programming interfaces to the sensors) support for position acquisition. Thus, developing a position-aware application is natural for a developer.

Let us step back and ask if position can adequately represent context in a personal agent. Before we answer this question, let us enumerate the typical ways in which a personal agent can employ context:

(1) *Explicit.* Using the information as is, as below:

- *Informative.* The agent can provide a user's context information explicitly, e.g., it may display the location in a calendar or tag location on a photo.
- *Social disclosure.* The agent can disclose a user's context to the social contacts of the user, e.g., on a social network site or in a text message.
- *Commercial disclosure.* The agent can disclose a user's context to a third party for

a commercial purpose, e.g., to obtain a coupon for a nearby coffee shop.

(2) *Implicit.* The agent can automate a task based on the user's context. For example, consider a personalization task such as changing the ringer mode of the user's phone or forwarding text messages to email, which can be performed automatically depending upon where the user is.

(3) *Prediction.* The agent can analyze a user's location (typically, location history) to discover interesting patterns and predict a future location. The future location could then be used for one of the above purposes.

Whether an agent employs context for the user to benefit from the location personally (for information or task execution), or exploits context to share it with others (for social or commercial purpose), what is a desirable level of abstraction at which to do so? We doubt it would be position; spatial coordinates do not have an inherent meaning for the user. Instead, we imagine that a notion such as *home*, *office*, *restaurant*, and *park* is more natural. We term this level of location abstraction *context*. Employing context instead of position has three implications for personal agents.

- By presenting context in a way that is natural to users, a personal agent can enhance *usability*, i.e., the ease with which a user can exercise an application [98].
- Context opens up new avenues for intelligent agents including personal assistants [19], pervasive and social games [71], recommender systems [124], and virtual worlds [45].
- Context can enhance *location privacy* [22] by providing users an easier means for controlling the extent to which their location information is shared, e.g., by sharing the information that a user is in a *class* instead of sharing the physical location of the specific class.

3.1 A Conceptual Metamodel

We restrict the generic notion of context to a specific abstraction called *place*. In contrast to context, which is defined as "any relevant information" [20], we base place on three contextual attributes: space, user activities, and social circles. In doing so, we make explicit what is that we seek to compute (recognize) and the corresponding assumptions.

The notion of place has been studied under constructs such as place attachment, place identity, sense of place, and semantic location. [32] identifies geographic location, material form, and meaning and value as three features of a place. [101] describe the meaning of a place using a tripartite model involving people (individuals or groups), place characteristics (social or physical), and processes (behavior, cognition, or affect). The interactionist theory of place attachment suggests that the meaning given by an individual to a physical site comprises the individual's memories of interactions associated with that site (interactional past) as well as future experiences perceived as likely (interactional future) [33, 64, 77]. [42] distinguish space and place by a phrase that:

> *"we are located in space, but we act in place."*

We synthesize (as [66] advocates) various place-related constructs as an informational entity that can be computed and employed in mobile applications. Figure 3.1 shows our conceptual metamodel, which can be used to model each place of interest to a user via its relationships with one or more of the following entities.

- *Space.* The spatial aspects of a place include one or more positions and the environment, including physical artifacts such as console and TV, and physical characteristics such as noise level, ambient light, and temperature.

- *Activity.* A place derives its meaning from the activities that occur there. For example, a user's *home* might be a place of entertainment, rest, and eating, whereas a *research lab* might be associated with activities such as writing a paper. Thus, a set of activities can be used to specify a place of interest to a user.

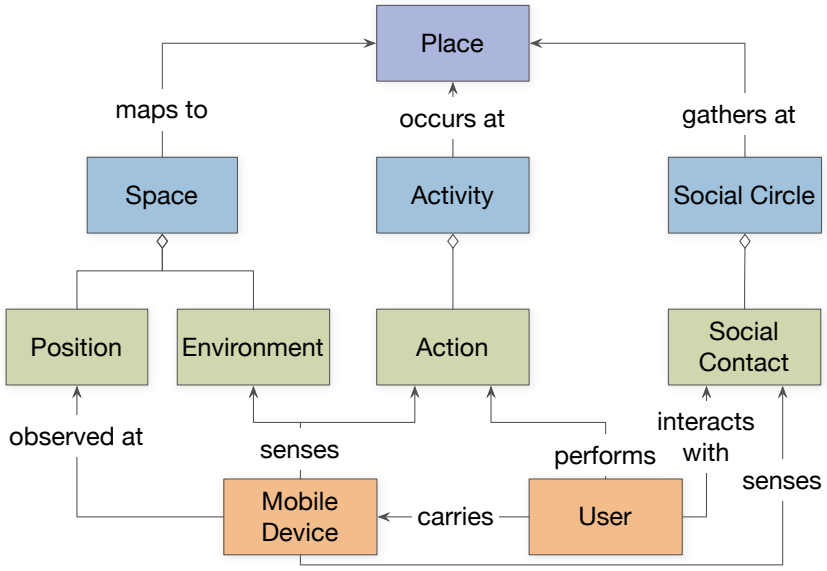


Figure 3.1: A conceptual metamodel relating place to space, activities, and social circles.

- *Social circle.* The places of interest to a user are often occupied by his or her social contacts. The user is likely to perceive a logical group of such contacts as a social circle [85]. For example, *home* is occupied by family members, *workplace* by colleagues, and *classrooms* by classmates. Thus, a place can also be described by the social circle associated with it.

We make three assumptions about modeling of places.

(1) A place can be completely specified by any combination of space, activity, and social circles. This assumption opens up interesting possibilities for spatially overlapping, dispersed, and space-less places. For example, two *classrooms* in different corners of a college campus can be the same place specified by the unique set of activities that take place in a classroom; the same *coffee shop* may be two different places—a *caffeine fix* and a *meeting place*—differentiated by the social circles involved; an Internet *chat room* might have no spatial aspects, but can be specified via activities or social circles.

(2) A places is ego-centric, e.g., *workplace* of a physician and that of a software engineer can

each be modeled as comprising different sets of activities.

(3) Places can be computed from space, activities, and social circles. Typical spatial aspects such as the position, temperature, and noise level can be sensed directly from a user's device. Activities and social circles can be computed from observable actions and interactions of a user. Examples of observable actions include URLs visited, applications used, and physical movement, and that of observable interactions include emails, text messages, and phone calls.

3.2 Challenges in Engineering Personal Agents

Although place offers potential benefits as a location abstraction, engineering a place-aware personal agent (PPA) is quite challenging. On the one hand, a PPA developer may target multiple users. Yet, the developer must engineer the PPA to deliver personalized services. On the other hand, a user may employ multiple PPAs, potentially each developed by a different developer. Yet, the PPAs (potentially complimentary) must deliver place-based services in a manner that maximizes user experience.

Providing a Channel between Developers and Users: How can a PPA developer model (subjectively) places a user may care for? In general, developers cannot determine the needs of each potential user. For example, a model yielding *home*, *office*, and *elsewhere* might suffice for a user, but another user might want a model that distinguishes multiple offices. Additionally, a user's location needs often change over time.

Providing a Channel among Developers: Each developer may employ a distinct place model imposing an unnecessary burden the user. For example, an application may model a *class* to include regular lectures and guest lectures whereas another application may differentiate the two events as taking place in a *lecture hall* and a *seminar hall*, respectively.

Reasoning about Places: Although places are high-level constructs, data for reasoning about places comes from low-level sensors. How can we map low-level sensor data to places? On the one hand, unsupervised techniques, requiring no labeling from users, cannot recognize places subjectively. On the other hand, supervised techniques require several training labels per place (for each user), imposing a huge labeling burden on users.

3.3 Platys: A Framework for Place-Aware Application Development

As shown in Figure 3.2, the Platys framework [83, 88] consists of sensors, a middleware, and applications. The middleware is its key component. In a nutshell, (1) a user interacts with the middleware and trains it about places of interest, (2) the middleware learns to recognize places of interest to each user from low-level sensor data, and (3) multiple applications interact with the middleware to know the user's places.

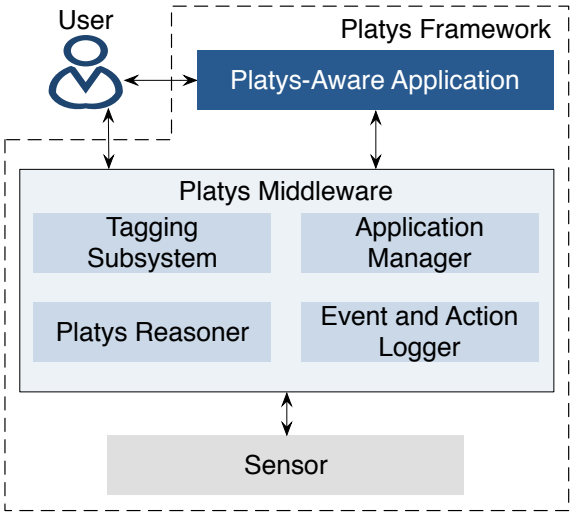


Figure 3.2: Platys framework consists of a middleware, sensors, and applications.

3.3.1 Platys Middleware

Let us consider the benefits of a middleware. We hypothesize that developing location-aware application employing a high-level abstraction such as place can be time-consuming. Thus, providing an off-the-shelf component that simplifies place-aware application development can be valuable. However, networking, coordination, delegation, and heterogeneity [26, 54] are inevitable requirements for building such a component because (1) data for reasoning about a user's places come from sensors on multiple devices, e.g., smart phone, tablets, and an increasing variety of wearable devices, and (2) the sensors, place reasoner, and place-aware applications may all reside on different hosts.

The Platys middleware is responsible for (1) efficiently gathering data from multiple low-level sensors; (2) computing high-level concepts such as places, activities, and social circles from low-level data specific to each user; and (3) exposing the learned high-level concepts to place-aware applications as per a user's needs.

Figure 3.3 shows the architecture of the Platys middleware consisting of four subsystems. The figure also shows the platform for which we have implemented each components. Each subsystem may be potentially hosted on any of a user's personal device. The subsystems communicate asynchronously via a shared information store.

The event and action logger aggregates data from multiple sources. Smart phones and wearable devices are ideal for hosting this component since they are equipped with sensors such as GPS sensors, accelerometer, gyroscope, Bluetooth, WiFi, camera, and microphone. In addition, the logger collects data from sources such as a user's call log, browsing history, email, SMS, and calendar. A user can control what sensors and data sources to use and at what frequency to collect the data.

For Platys to make sense of the sensor data, the tagging subsystem helps a user train Platys on the relevant place, activity, and social circle. Since smart phones are always with a user, they are ideal to deliver notifications prompting the user to tag. Figure 3.4b shows the user interface from our Android implementation of the tagging subsystem. The user may ignore any prompt

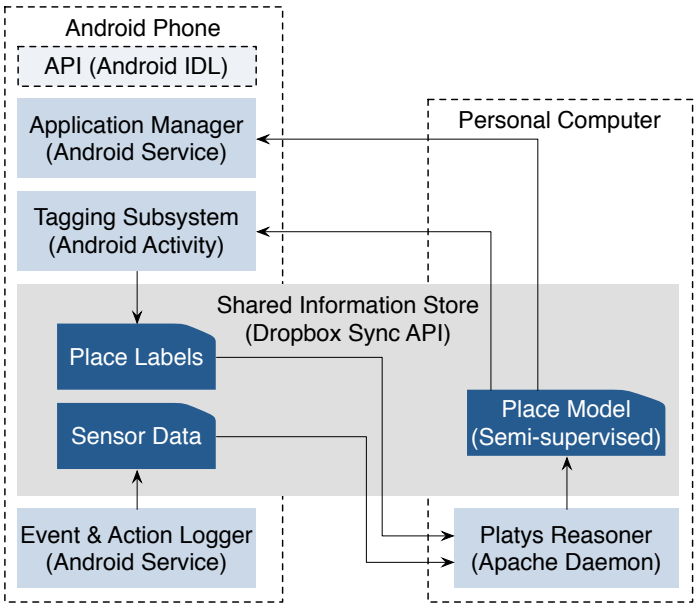
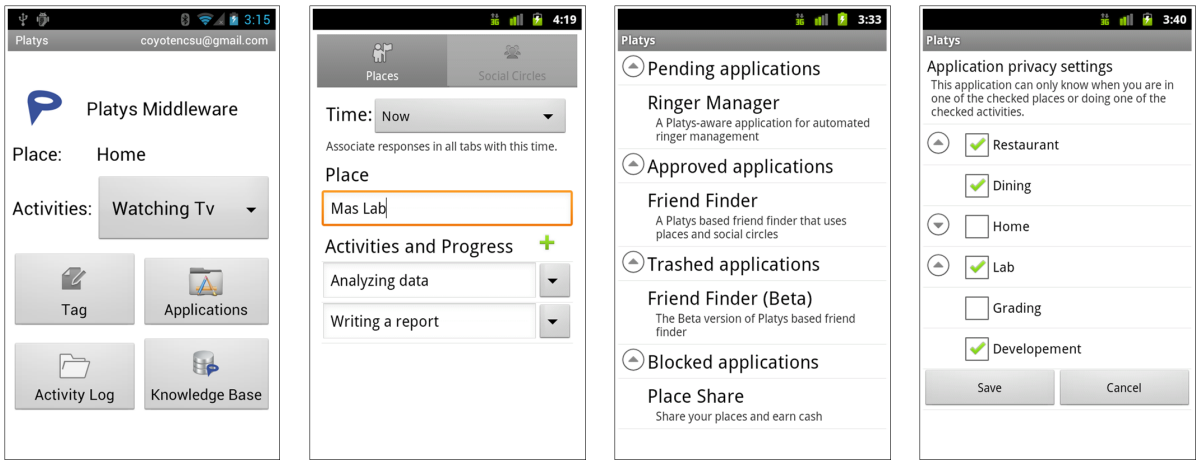


Figure 3.3: Platys middleware's subsystems. The subsystems are loosely coupled and communicate asynchronously via a shared information store. Each of a user's personal devices can host one or more of the subsystems.

or delay responding.

The Platys reasoner builds a machine learning model to associate user tags with sensor data. In a typical scenario, the user tags places for a training period and the reasoner subsequently predicts the places, activities and social circles. Further, the reasoner assigns a confidence level to its predictions in order to enable active learning. A resource-rich device such as a user's personal computer (compared to a mobile device) is ideal for hosting the reasoner.

Platys-aware applications interact with the application manager to acquire a user's places, activities, and social circles. The application manager respects privacy preferences specified by a user as shown in Figures 3.4c and 3.4d. An instance of the application manager must be hosted on each device which hosts place-aware applications.



(a) The home screen shows the reasoner's predictions about a user's current place.

(b) A user tags the current or a recently past place, activities, or social circles.

(c) The user can approve, trash, or block a pending application from here.

(d) The user can set fine-grained privacy policies for a Platys application from here.

Figure 3.4: Screenshots from the Platys middleware's subsystems.

3.3.2 Platys-Aware Application Development

A typical workflow of how a user, a Platys-aware application, and the Platys middleware interact with each other is shown in Figure 3.5. Platys-aware application development tackles the challenges discussed in Chapter 1. First, it provides a communication channel between application developers and users. With Platys, *what users tag is what an application can see*: A user trains the Platys middleware to recognize places of interest; the Platys middleware learns to recognize the tagged places and exposes them to applications. Thus, an application can rely on the Platys to provide places of interest to each of its users. Further, a user can tag new places and Platys automatically updates the places it exposes to the applications.

Second, Platys exposes places uniformly across all applications. Since each Platys-aware application employs places exposed by Platys, the user avoids the burden of understanding multiple place models.

Third, Platys provides users fine-grained control on privacy. In current practice, a user's control on privacy is typically at the level of all locations or none. However, a user's willingness

to disclose location depends on who is requesting, why, and the details requested [110]. Platys supports such fine-grained privacy policies.

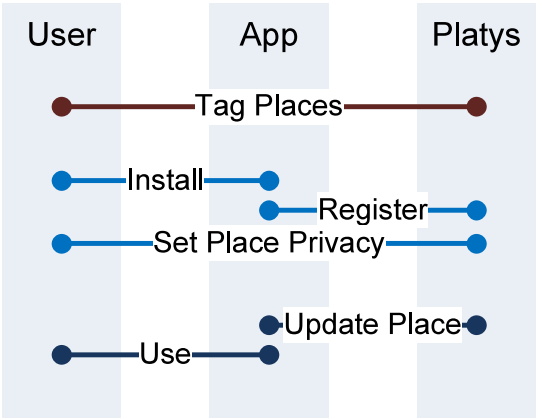


Figure 3.5: Interactions between a user, a Platys-aware application, and the Platys middleware. The user tags places on an ongoing basis; the application is installed and registered; the middleware continually sends place updates to the application as needed.

3.4 Developer Study

We now evaluate Platys as application development platform. We evaluate the platform from the perspectives of two kinds of stakeholder—application developers and end-users. We evaluate the:

(1) *efficiency* of the middleware in assisting developers with respect to time and effort, and

(2) *effectiveness* of the middleware in enabling high-quality place-aware applications from both the developer and end-user perspectives.

3.4.1 Study Design

Our study design terminology is adapted from [55]. For convenience, we summarize the key terms (with their mapping to our study) in Table 3.1.

Table 3.1: A brief description of the study-design terminology we adopt.

Term	**Description**	**Examples in our study**
Study unit	An object on which the study is being conducted.	A location-aware application.
Subject	A participant in the study.	A developer exercising an approach.
Parameter	A characteristic held invariant throughout the study.	Complexity of the software to be developed.
Response variable	A variable measuring the outcome of a study.	Usability of the product, development time.
Factor	A characteristic studied that affects a response variable	Development platform
Alternatives	The different values of a factor studied.	Android location manager vs. Platys middleware.
Undesired variation	A characteristic that we wish to keep invariant, but cannot.	Programming experience of the subjects.

Study Unit

The unit of our study was the location-aware application to be developed. We conceived an application called the *Ringer Manager Service (RMS)* that automatically sets the ringer mode of a user's mobile phone based on location. The functional requirements of the application were the following.

- RMS must continually monitor a user's location.
- RMS must provide a user a means of assigning a ringer mode (loud, vibrate, or silent) to locations of interest.
- RMS must automatically adjust the ringer mode of a user's phone according to the user's setting for the current location, setting it to a default if the user has not specified a ringer mode for the current location.
- RMS must also act as a notification manager in the scenario when a user's phone is

in *silent* mode and the user misses a call by sending a notification to the caller. The notification should contain location information, e.g., "Sorry for missing your call; I am in a lecture hall right now."

So that it is representative of a variety of location-aware applications, the RMS was designed to use location for multiple purposes—informative, task execution, and social disclosure. In addition to the functional requirements, we also specified a set of requirements to enhance the usability and privacy of RMS:

- RMS should be able to capture ringer mode settings for as many locations of interest to a user as possible.
- RMS should accommodate the changing location needs of a user.
- RMS should equip users with utmost control on privacy.
- RMS should be usable by a variety of users. Thus, developers should avoid making assumptions that won't generalize to a wide variety of users.

The usability requirements were specified at a fairly high level to encourage developers to use their natural intuitions in addressing them. Note that a generic notion of usability of a mobile application depends on several factors. Our focus here was to evaluate the usability of RMS specific to its location-aware aspects.

Next, we divided the experimental unit into four subunits. Each subunit represented an essential step in the development of RMS.

(1) *Preparation (prep).* Setting up the development environment, familiarizing with the application specification, and acquiring the necessary background knowledge.

(2) *Location representation and acquisition (loc).* Representing the location at a suitable level of abstraction and developing techniques for acquiring it.

(3) *Core functionality (core).* Implementing the functionality of (a) providing users an option to set the ringer mode, (b) automatically changing the ringer mode based on the location, and (c) sending a notification to a caller on a missed call when the phone is silent.

(4) *Usability and privacy (usability).* Enhancing the usability and privacy of RMS.

Subjects

Our study involved 46 students enrolled in a graduate-level computer science course (36 graduate, four undergraduate, and six online graduate; 27 male and 19 female). The study was approved by our university's IRB. Subjects earned points (counting toward the course grade) for completing the study. However, participation in the study was not mandatory. Nonparticipants were offered an alternative task to earn points equivalent to what they would earn by participating in the study.

Study Mechanics

We asked each subject to develop RMS from the functional and usability requirements. In addition to developing the application, subjects were asked to keep track of the time and effort they expended for development by answering a *time and effort survey* after each development session. The survey asked what subtasks each subject worked on during a session, how long he or she spent on each of those subtasks, and how difficult he or she felt a particular subtask was. The subjects reported time in hours and minutes, and difficulty on a scale of *very easy*, *easy*, *medium*, *difficult*, and *very difficult*. Finally, the subjects were asked to produce a document describing how they addressed the usability requirements in their application.

Parameters

We identified the following as the parameters of the study.

- *Requirements:* For uniformity, all subjects were given both functional and usability requirements, which remained unchanged, for the most part, during the study. Minor

changes and clarifications were announced via a website and all subjects notified via email.

- *Deliverables:* The deliverables of the project were the same for all subjects: time and effort surveys, source code of the project, and a document describing the usability and privacy-enhancing features of the application.

- *Study duration:* All subjects were given a four-week period to submit all deliverables (we allowed one additional week for one subject for medical reasons).

- *Software tools:* All subjects were required to use Eclipse 3.6+ as the development platform and Android Development Tools (ADT) plug-in for developing Android applications at API level 10.

- *Development device:* Each subject was provided with an Android development phone for the duration of the study unless he or she opted to develop on a suitable personal Android device.

Response Variables

Table 3.2 summarizes the responses variables we analyzed. For each subject, the $time_{subtask}$ was calculated as the sum of times reported by the subject for the *subtask* across multiple sessions. Here, $effort_{subtask}$ is the arithmetic mean of effort ratings reported by the subject for the *subtask* across multiple sessions.

We analyzed the overall time and effort required to develop RMS from two perspectives—including and excluding the preparation time. We defined the following variables: $time_{RMS}$ as the sum of $time_{subtask}$ for each *subtask*; $effort_{RMS}$ as the arithmetic mean of $effort_{subtask}$ for each *subtask*; and $time_{RMS-prep}$ and $effort_{RMS-prep}$ as above but respectively excluding $time_{prep}$ and $effort_{prep}$. The motivation was to compare subjects in terms of their experience in location-aware application development. A developer needs to perform the preparation subtask only for the first location-aware application he or she develops. Since most of our subjects

Table 3.2: A description of the response variables we analyzed. The *subtask* variable in the table can take values *prep*, *loc*, *core*, and *usability*.

Response variable	**Study unit**	**Description**
$time_{subtask}$	*subtask*	Development time reported by subjects.
$time_{RMS}$	RMS	
$time_{RMS-prep}$	RMS after *prep*	
$effort_{subtask}$	*subtask*	Perceived effort as reported by subjects.
$effort_{RMS}$	RMS	
$effort_{RMS-prep}$	RMS after *prep*	
MCC	RMS	McCabe's cyclomatic complexity.
$NoLM$		Number of levels per method.
NoS		Number of statements.
NoM		Number of methods (for a fixed NoS).
usability	RMS	Extent to which an RMS implementation meets usability and privacy requirements.

were inexperienced in location-aware development, we used $time_{RMS-prep}$ and $effort_{RMS-prep}$ as indicators of the time and effort expended by experienced location-aware developers.

Next, we analyzed the quality of the applications produced from two perspectives.

(1) *Developers.* We employed the following well-known software metrics [76, 93] as indicators of the quality of the software modules produced.

- MCC: McCabe's cyclomatic complexity indicates the number of "linear" segments (i.e., sections of code with no branches) in a code fragment. This is an indicator of the psychological complexity of a code fragment. We measured the MCC of the project as the mean of MCC for each method in the project.
- $NoLM$: The number of levels per method reflects the number of logical branches each method has on average. The metric is a key indicator of code readability. We measured the $NoLM$ of the project as the mean of the $NoLM$ for each method in the project.

- NoS: The number of statements in a project is an indicator of the general maintainability of the code. We measured NoS as the sum of non-comment and non-blank lines inside method bodies of a project.
- NoM: The number of methods in a project (for a fixed NoS) is an indicator of the modularity of the code.

(2) *End-users.* We performed a qualitative analysis of each application (end product) produced. The objective of the analysis was to understand the techniques employed by each subject to meet the usability and privacy requirements outlined earlier. The techniques employed by an application allude to the potential usability problems associated with the application.

Factors and Alternatives

Our objective was to study the effect of the location abstraction employed—position or place. The abstraction a developer employs depends on the location acquisition platform available. We divided subjects into two equal sized groups as follows. The *Control Group* employed the Android SDK [3] for location acquisition, which provides position information from GPS or the network. The *Platys Group* employed the Platys middleware as [80] for location acquisition platform, which provides place information.

The choice of Android SDK as the alternative of Platys is motivated by two factors. First, the Android SDK is the de facto standard platform for developing location-aware Android applications. Thus, our findings could be of interest to a large developer community. Second, although platforms with similar objectives as Platys are described in the literature (reviewed in Section 3.5), none are available for easy deployment on the Android platform to enable a fair comparison with Platys.

Undesired Variations

We identified three sources of undesired variation and sought to mitigate the associated risks as follows.

- *Subjects' experience:* Differences among subjects' programming experiences is inevitable in our setting. A subject's programming experience can influence the time and effort he or she expends on a programming task. To minimize the risks associated with the difference in subjects' skill sets, we conducted a prestudy survey asking subjects about their experience in general, Android, and location-aware programming. We assigned approximately an equal number of subjects at each level of experience to the Control and the Platys Groups. Assignment within each level of experience was completely random, though. However, most of our subjects (86%) were new to developing mobile or location-based applications. Thus, each subject was required to complete a simple location-based Android programming exercise prior to the study to acquire basic knowledge of Android programming.

- *Communication between subjects:* We noticed that communication among subjects across the Control and Platys Groups could influence a subject's (a) strategies for enhancing usability and privacy of RMS, and (b) survey responses to the perceived effort, if the subject figured out whether he or she belonged to Platys or Control Group. In order to minimize the risks associated with this factor, the groups were called Group 1 and Group 2. Further, we strongly discouraged subjects from communicating with each other about the task. All communication between the subject and the researchers were through one-to-one channels (email or meetings) instead of a message board. Although the requirements for both groups were the same, we provided group-specific guidelines accessible only to the group members.

- *Different levels of formalization:* The level of formalism and the resources available for developing Platys-aware applications is small compared to that of developing position-

aware applications with Android SDK. Although we exposed a descriptive API and sample programs for Platys, minor changes to the API were inevitable during the study, especially in the early parts.

3.4.2 Analyses Performed

At the end of the study, we verified the submissions and found 12 submissions to be incomplete (seven from Control and five from Platys Group). An incomplete submission did not address each functional requirement. Our results are based on 34 complete submissions, which comprise of 16 Control and 18 Platys subjects. Our analyses considered the following statistics:

- mean of the sample for t-test;
- variance of the sample for F-test; and
- average rank of the sample for Wilcoxon's *ranksum*-test (typically, difference in average ranks of two samples indicate a difference in corresponding medians).

For each statistic, we tested the null hypothesis H_{Null} against the alternative hypothesis H_{Platys} or $H_{Neither}$ described in Table 3.3. We use Platys and Control *subjects* to refer to the two samples studied and Platys and Control *developers* to refer to the corresponding populations.

Table 3.3: Null and alternative hypotheses. Each test verified the null hypothesis (H_{Null}) against one of the alternative hypotheses (H_{Platys} or $H_{Neither}$).

ID	Hypothesis
H_{Null}	There is no difference in the *statistic* for Platys and Control developers.
H_{Platys}	The *statistic* for Platys developers is less than that for Control developers.
$H_{Neither}$	There is a difference in the *statistic* for Control and Platys developers.

All tests accommodated samples of unequal sizes. The t-tests were conducted assuming unequal variance between the two populations (also called Welch's t-test). For t-tests, we verified that the corresponding samples passed the Kolmogorov-Smirnov normality test. A one-tailed

or two-tailed test was conducted depending on whether the null hypothesis H_{Null} was tested against the alternative hypothesis H_{Platys} or $H_{Neither}$, respectively [48, 31].

3.4.3 Results and Discussion: Time and Effort

Figures 3.6 and 3.7 compare the development times and effort ratings reported by Control and Platys subjects during the development of RMS. We also summarize the results of hypothesis testing in the figure (to the right of each plot). We compared difference in means (μ) and variances (σ^2) for times reported, and median ($\widetilde{x}$) for effort ratings. Comparisons involving $<$ and $\neq$ indicate the alternative hypotheses H_{Platys} and $H_{Neither}$, respectively. Highlighted are the significant differences ($**$ and $*$ indicate sufficient evidence to reject the null hypothesis at 5% and 10% significance levels, respectively).

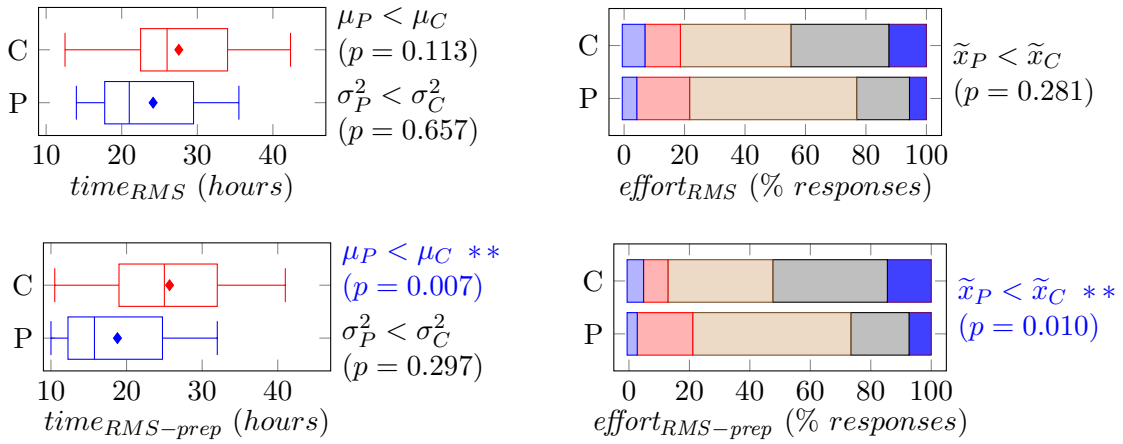


Figure 3.6: Comparison of the time (left) and effort expended (right) by Platys (P) and Control (C) subjects to develop RMS, highlighting significant differences.

We now discuss the motivations behind our hypotheses and whether the observations supported our hypotheses. In case of inconsistencies, we discuss if an undesired variation could have influenced the result.

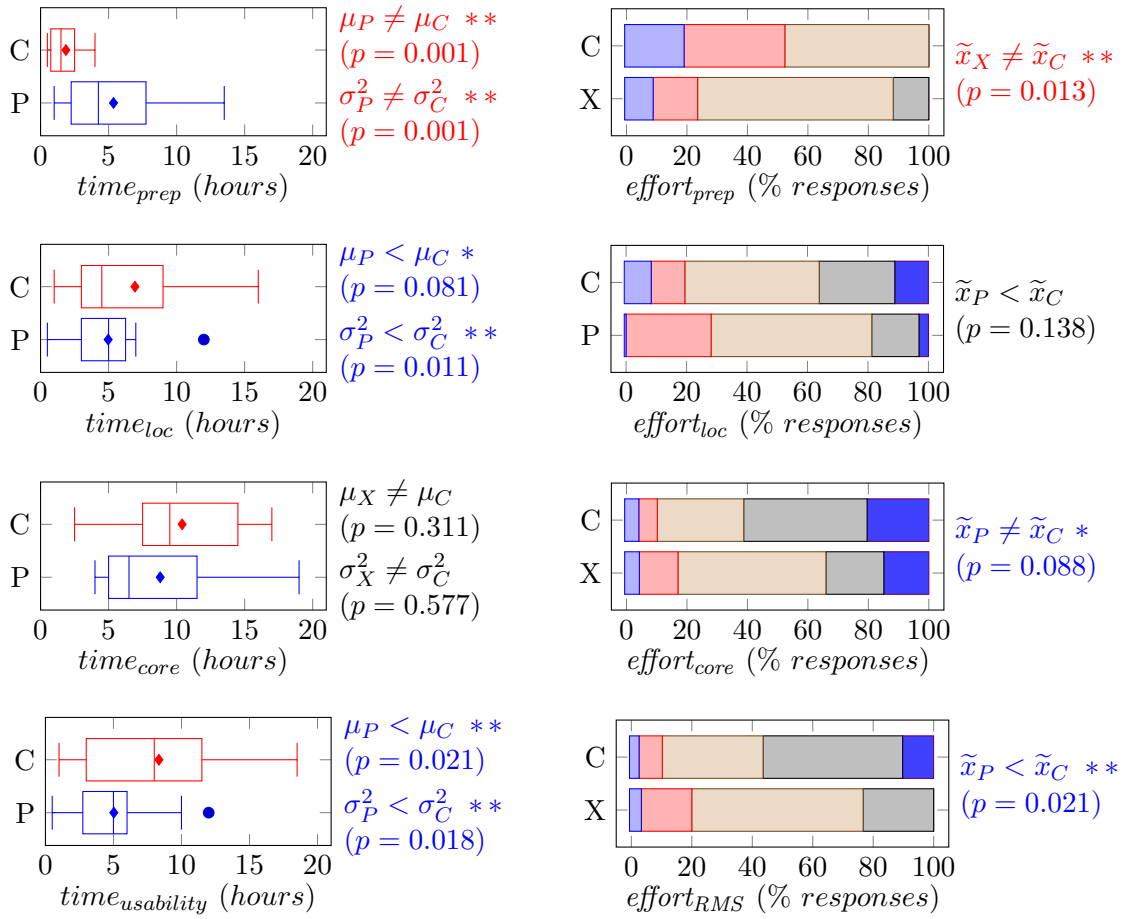


Figure 3.7: Comparison of the time (left) and effort expended (right) by Platys (P) and Control (C) subjects to develop RMS' subtasks, highlighting significant differences.

Preparation

As part of the preparation for RMS development, each developer must become familiar with the functional and usability specifications of RMS, and set up the development environment. Other than these steps, the only other task for a Control developer is to become familiar with the Android location API. However, a Platys developer must install the Platys middleware, and become familiar with both the Platys place API and AIDL (Android Interface Definition Language) to interact with the middleware.

Clearly, a Platys developer must perform more preparatory work than a Control devel-

oper. Thus, we hypothesized that Platys developers would expend more time and effort for preparation ($H_{Control}$). Not surprisingly, the observations supported our hypothesis. The difference in variances was not surprising, either, considering the fact that Control developers have noticeably less preparatory work to do. However, these results are not discouraging. The important question is whether the extra cost expended by Platys developers for preparation pays off elsewhere.

Location representation and acquisition

The simplest approach for a Control developer to do is to represent location as position and acquire position information from the Android location API. For a Control developer who wishes to abstract location at a higher level than position, the location representation and acquisition time is likely to be high. For a Platys developer who is acquainted with the Platys middleware, representation and acquisition come at a low cost. That is, a Platys developer can represent location as place and acquire place by interacting with the Platys middleware. Thus, we hypothesized that Platys developers would spend less time and effort for representing and acquiring location (H_{Platys}). The difference in the means and variances of the times reported by Control and Platys subjects supported our hypothesis.

Further, we observed that all Platys subjects represented location as place, whereas Control subjects used a variety of techniques to represent location. We summarize the major techniques employed by Control subjects to represent location below.

- A pair of spatial coordinates with a fixed radius for each location. Four subjects implemented this technique (25%).
- A pair of spatial coordinates with a configurable radius for each location. Two subjects implemented this technique (12.5%).
- A conceptual unit (i.e., a location with a logical name) backed by a pair of spatial coordinates. However, the list of conceptual units is preconfigured by the developer, e.g., one

of the preconfigured list was *home* and *office*. Three subjects implemented this technique (18.75%).

- A conceptual unit backed by a pair of spatial coordinates and a user can add any number of conceptual units. Seven subjects implemented this technique (43.75%).

We noticed that 75% of the Control subjects attempted to represent location at an abstraction higher than position. This explains why Control subjects spent significant amounts of time for representing and acquiring location and points toward the need for architectural support to represent and acquire location as a high-level abstraction.

Further, there was insufficient evidence to reject the null hypothesis for difference in the effort ratings (although the median effort rating for Platys subjects was smaller). This outcome could be explained by the different levels of formalization between the Platys and the Android APIs. Since the Platys middleware is not a commercial product, we encountered unanticipated patterns of middleware usage from the subjects, which required minor changes to the middleware in early stages of the study. Working with a middleware that changed, albeit slightly, might have made development more difficult for Platys subjects.

Core functionality

Given that developers who have already represented location know how to acquire it at the desired level of abstraction, the core functionality to be implemented by Control and Platys developers is the same. Thus, we hypothesized that there is no difference in the time and effort expended by Control and Platys developers for implementing the core functionality ($H_{Neither}$).

The results pleasantly surprised us. Although the difference in times reported were not significant between Control and Platys subjects ($p = 0.311$), the efforts reported by Platys subjects were significantly less than those of Control subjects ($p = 0.088$). This leads us to conjecture (for future study) that a better representation of location can lead to reduced effort in implementing the core functionality.

Usability

The Platys middleware seeks to support usable location-aware applications. To assist developers in enhancing the usability and privacy of a location-aware application, the Platys middleware provides developers with access to places and activities, social circles, and privacy policies. Further, the Platys middleware notifies applications of newly added and stale places so that they can adapt to the changing location needs of a user. However, for a Control developer, incorporating such features involves a nontrivial investment of time and effort. Thus, we hypothesized that Platys developers would spend less time and effort for enhancing usability than Control developers (H_{Platys}). The observations supported our hypotheses for the difference in mean and variance of times reported as well as the difference in median effort expended. In each case, the null hypothesis was rejected at a significance level of about 2%.

Ringer Management Service

From the perspective of inexperienced location-aware developers, Platys developers would spend extra time and effort in preparation but that expense would pay off in representation and acquisition, and enhancing usability. Thus, we hypothesized that Platys developers would do at least as well as Control developers, if not any better ($H_{Neither}$). The observations supported our hypothesis. Further, the p-values obtained indicate that the time and effort expended by Platys developers would be smaller (although not significantly).

From the perspective of experienced location-aware developers, Platys developers gain some advantages over Control developers. Thus, we hypothesized that Platys developers would do better than Control developers in time spent and effort expended (H_{Platys}). The observations supported our hypotheses about mean time and median effort at about 1% significance level.

However, the observations didn't support our hypothesis that the variance in time reported would be smaller for Platys developers than Control developers (for both inexperienced and experienced developers). Further investigation revealed that a significant amount of variance in times reported by Platys subjects originates from the variance in times for the core functionality

task. Such variance could arise because of the fact that subjects had no incentive to submit the deliverables early. The Platys subjects would have spent more time on the core task while Control subjects spent some of their time on other aspects of the project.

3.4.4 Results and Discussion: Software Metrics

In this and the next section, we analyze the quality of the applications produced. In order to understand quality from developers' perspective, we analyzed well-known software metrics (computed from the source code of the applications developed by subjects). Because place is a high-level abstraction and the Platys middleware supports representing and reasoning about place, we hypothesized that applications (software modules) produced by Platys developers are easier to comprehend (MCC), easier to read ($NoLM$), shorter (NoS), and more modular (NoM), i.e., H_{Platys} for each software metric we analyzed.

The boxplots in Figure 3.8 compare the software metrics of the RMS applications developed by Control and Platys subjects (computed from the source code). The figure also summarizes the results of hypothesis testing for each metric. Our observations support H_{Platys} for both MCC and $NoLM$ at less than 1% significance level. This indicates that applications developed by Control developers, who employ position abstraction, are likely to be harder to comprehend than those developed by Platys developers.

However, the results were not according to our intuition for NoS and NoM. Although each metric was slightly better for Platys subjects, the evidence was not significant to reject the null hypothesis (at 5% significance level) in either case. It was surprising that the amount of code produced was not significantly different across groups although Control subjects spent more time in doing so than Platys subjects.

An analysis of the variance (also summarized in Figure 3.8) revealed that the variance in NoS for Control subjects was significantly higher than that for Platys subjects. This variance could result from the varying extents to which Control implementations met usability requirements (since the analyzed applications meet all functional requirements) We conjecture that

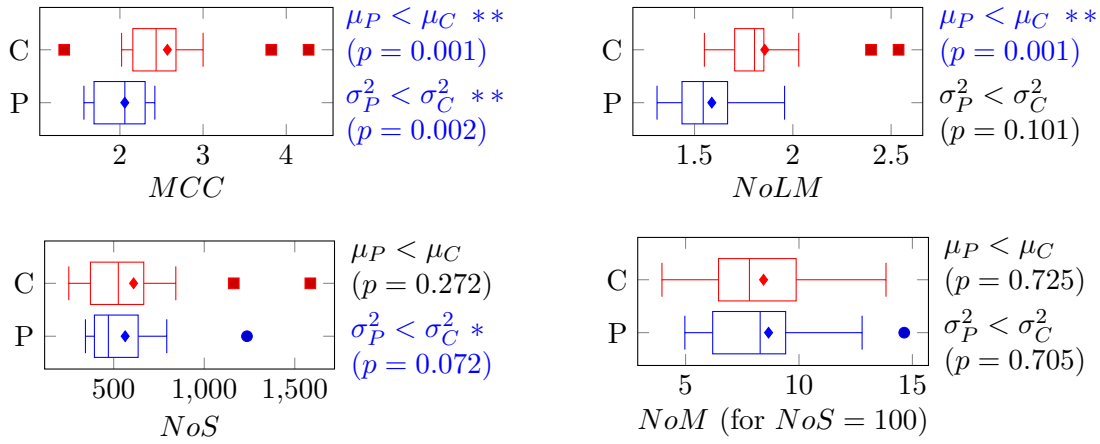


Figure 3.8: Comparison of the software code metrics for the RMS implementations produced by the Platys (P) and Control (C) subjects. Also shown are the results of hypothesis testing, highlighting significant differences (to the right of each plot).

the mean code size of Control implementations would be higher if all Control implementations met all usability requirements (Section 3.4.5).

Our results about NoM for code sizes are inconclusive. RMS implementations of Control subjects were at least as modular as those by Platys subjects. However, whether this would continue to hold had all Control subjects addressed usability requirements effectively (which would increase the code sizes) remains to be verified.

3.4.5 Results and Discussion: Usability

In order to understand quality from end-users' perspective, we performed a qualitative evaluation of the usability of RMS applications developed by Control and Platys subjects. To do so, we analyzed the usability description document submitted by each subject and verified the claimed features by testing the subject's application. In the process we discovered features that were not claimed, but implemented, which could potentially affect the usability of RMS. Table 3.4 summarizes, in three categories, major techniques employed by subjects to address the usability requirements. Next, we discuss potential impact these techniques may have on the usability of RMS.

Table 3.4: A summary of the techniques implemented by Control and Platys subjects to address the usability requirements.

Category	Technique Implemented	Control (% sub.)	Platys (% sub.)
Visualization	Logical names as a list.	56.25	77.78
	Logical names on a map.	6.25	22.22
	Unlabeled markers on a map.	25.00	0.00
	Spatial coordinates as a list.	12.50	0.00
Evolution	Notify new and stale locations automatically.	0.00	83.33
	Users manually add new locations.	81.25	0.00
	No support for evolution.	18.75	16.77
Privacy	Specify a policy for each social circles.	0.00	77.78
	Specify a policy for each contact.	18.75	0.00
	Share with anyone in the contact list.	6.25	0.00
	Ask user each time before sharing.	62.50	11.11
	Specify to share with all or none.	12.50	11.11

Visualization

An RMS implementation must display locations of interest to a user for informative purposes, e.g., for showing ringer modes associated with locations. Most Platys subjects showed locations as a list of previously tagged places (and some marked the place on a map when clicked). Note that showing places on map is not always a viable option for Platys-based RMS implementations since (1) not all places may have a spatial component; (2) a user may configure the Platys middleware to not share spatial coordinates with RMS at all.

Interestingly, RMS implementations of more than half of the Control subjects also visualized location as logical names (as a list or on a map). However, techniques implemented by other Control subjects could potentially reduce usability, e.g., both unlabeled markers and the list of spatial coordinates reduce the *memorability* of the user interface and a list of spatial coordinates may not be *intelligible*.

Evolution

As a user visits different locations, RMS should enable the user to set (or reset) an appropriate ringer mode for each location. The Platys middleware notifies registered applications of new locations tagged by the user as well as locations that have become stale. Most Platys subjects implemented RMS so as to take advantage of these notifications and prompt the user to add (or delete) ringer modes for new (or stale) locations. A few Platys subjects ignored these notifications and didn't address the requirement of evolving RMS as the location needs of a user change.

The RMS implementations by most Control subjects provided a user an option to manually add locations as needed, but only 37.5% of them provided an option to delete stale locations. Requiring the user to manually add each location can be *time consuming* (as opposed to automated support). Further, being unable to delete stale locations can easily *clutter* the user interface. The rest of the Control subjects' implementations preconfigured the list of locations (e.g., *home*, *office*, and *restaurant*) allowing a user to neither add nor delete locations. Such preconfigured lists don't necessarily *generalize* to a variety of RMS users.

Privacy

The Platys middleware provides a user an option to specify which of the user's locations are to be shared and with whom (one or more social circles). A majority of Platys subjects implemented RMS to consult the Platys middleware (through appropriate method calls) before sharing the location with a caller. In contrast, a majority of the RMS implementations by Control subjects consulted the user before sharing location. Although this option is privacy preserving, it is too *intrusive*. Asking a user each time before sharing location defeats the very purpose of RMS to automatically notify callers. The other options implemented by Control subjects were also suboptimal: specifying a policy for each contact is *time consuming*, and automatically sharing location with anyone in the contact list is too *coarse*. Finally, none of the RMS implementations by Control subjects enabled a user to specify the *granularity* at which location is to be shared

(e.g., logical names only, include spatial coordinates, and so on).

For Platys subjects, addressing the usability requirements, for the most part, was a matter of employing the Platys API appropriately. As indicated above, a majority of Platys subjects succeeded in doing so. Control subjects attempted to address the usability requirements in a variety of ways. However, many of the techniques implemented by Control subjects could potentially impact the usability negatively. The shortcomings outlined above indicate that despite the extra time and effort expended, the usability enhancing features implemented by Control subjects were not as effective as those implemented by Platys subjects.

3.5 Related Work

3.5.1 Place Models

To realize a conceptual model of location in a development environment to support a variety of location-aware applications is challenging. Ranganathan et al.'s [95] MiddleWhere middleware realizes a hierarchical model of location involving points, lines, and polygons backed by physical coordinates. Stevenson et al.'s [113] LOC8 framework realizes a model of location consisting of a granularity (coordinates or symbolic names) and spatial relationships (containment, adjacency, connectedness, and overlap). Ye et al. [132] describe additional implementations that realize space models. Approaches that model only space and spatial relationships fail to capture place in its entirety.

Baldauf et al. [6] survey several context-aware approaches. Bettini et al. [9] provide a comprehensive view of several context modeling and reasoning techniques. Although existing context-aware approaches model more than space, they largely focus on environmental features of context. Schuster et al. [103] survey several approaches that bringing together spatio-temporal aspects of context and online user interactions (e.g., on a social network site). These approaches capture only a fixed set of objective contexts.

The Platys middleware is novel in that it unifies space, activities, and social circles into

the notion of place. The three features of place are captured, not as independent entities, but in a unified manner. The middleware is extensible and captures places of interest to each user subjectively. Further, each user can control which of his places an application can access and at what granularity. Additionally, Platys promotes confidentiality and privacy by running locally on a user's personal devices.

3.5.2 Place-Aware Application Development

Below, we identify location and context-aware systems that include architectures, methodologies, programming frameworks, tools, and techniques.

CARISMA [14] is a context-aware reflective middleware that provides primitives to handle context changes using policies and to resolve conflicts that arise with them. The middleware is mainly evaluated for computational performance. Its usability is informally evaluated by a single subject. Capra et al. identify the need for studying the amount of work required by application engineers to develop a context-aware application as an important future work—this is what we study via Platys.

Topiary [67] is a prototyping tool for location-enhanced applications that seeks to enable interaction designers (with limited expertise on location acquisition techniques) to prototype location-aware applications. We envision a Topiary-like tool to be an extension of the Platys middleware that could receive context components from Platys (unlike the fixed, built-in, context components of Topiary). Li et al. informally evaluate Topiary observing that some of their subjects familiar with ubiquitous computing find the high-level abstractions of Topiary easier to deal with than sensors and logic-based rules. Whereas Li et al. evaluate the usability of the Topiary tool itself, we evaluate the usability of the location-aware applications produced from Platys.

LIME (Linda in a Mobile Environment) [79] consists of a coordination model and middleware for dealing with mobility of hosts and agents. The crux of LIME is the idea of transiently maintaining a tuple space of context data, which could potentially simplify application design.

The LIME middleware supports both private and grouped tuple spaces. In contrast, Platys maintains each user's place information privately. However, Platys could enable coordination at the level of social circles (compared to LIME groups which represent agents colocated on a host). LIME is evaluated informally through two case studies and presents results as "lessons learned."

TOTA (Tuples On The Air) [72] consists of a middleware and a programming approach. The middleware facilitates generation of context tuples by applications, and propagation and maintenance of such tuples according to application specific rules. A major objective of the middleware is to alleviate developers from dealing with low-level issues such as representing context and network dynamics. TOTA is evaluated via simulation for performance metrics such as the propagation time and number of maintenance operations required under various circumstances.

OPEN [37] is an ontology-based programming framework for prototyping context-aware applications. The major objective of OPEN is to cater to developers ranging from novices (e.g., as in end-user programming) to experts. Accordingly, OPEN consists of three programming modes. Its evaluation compares the programming modes with each other for relative accuracy and ease of use.

Hermes [13] is a context-aware application development toolkit that seeks to reduce the overhead of context-aware application development associated with sensing, aggregating, and inferencing context information. Hermes provides an intuitive description of how it could reduce the overhead, but no empirical evidence. Like Hermes, the Platys middleware is loosely coupled. However, the Platys middleware implements the place reasoner as one module whereas Hermes employs multiple widgets. We conjecture that a unified treatment of place enhances the intelligibility and simplifies design.

Kulkarni et al. [61] describe a programming framework for context-aware application development that requires an application developer to produce domain-specific models of an application in terms of policies regarding *activities*, *roles*, *objects*, and *reactions*. Next, a generic

middleware generates an execution environment consisting of specialized application-specific components. Kulkarni et al. evaluate for the efficiency (time required) of the generative process and report the number of automatically generated components (of testbed applications) as a potential indicator of the development work the middleware could reduce.

The works mentioned above seek to simplify location-aware or context-aware application development but do not evaluate the effectiveness for developers empirically. Instead, the evaluations consider metrics such as computational time. Although establishing such characteristics is important, equally important for engineering are the benefits the approaches offer to developers and end-users.

To the best of our knowledge, the Platys developer study reported in this paper is the first of its kind in that it quantifies the efficiency and effectiveness of a location-aware middleware from the perspectives of application developers and end-users. The study analyzes the implications (to developers) of employing the middleware at the granularity of the subtasks of the development process. Further, it highlights the potentially superior user experience place-aware applications could offer to an end-user. From our experience, we have learned that such studies are difficult to design, control, and conduct. The analyses we performed could be valuable to location-aware applications' researchers and developers alike.

3.6 Directions

We describe three directions in which our work can be extended.

3.6.1 Enhancing the Platys middleware

The Platys middleware can be extended in two ways.

(1) Two major concerns about place recognition are the user effort it demands and the battery power it consumes. Platys addresses the first concern via active learning. To address the second concern, Platys adopts an extreme solution by collecting sensor readings passively

(only when another application invokes a sensor). Whereas this approach conserves power, it may yield suboptimal place recognition accuracy. Platys can benefit from adaptive sensing techniques such as sensor suppression and substitution [140]. We defer the task of studying the tradeoff between place-recognition accuracy and power consumption to future work.

(2) Platys enables a user to specify fine-grained application-centric privacy policies. That is, a user can specify for each application, the places and the underlying attributes the application can access. In contrast, Tiwari et al. [120] describe a context-centric approach in which a user can specify "bubbles" of contextual events and applications that can execute within each bubble. The hierarchical place model that Platys builds can be used to construct bubbles described by Tiwari et al. However, the implications of application-centric and context-centric approaches on user experience remain to be studied.

3.6.2 Usability Evaluation of Place-Aware Applications

We performed a qualitative evaluation of the usability of RMS applications. However, Duh et al. [23] observe that several critical usability related problems can only be uncovered in a field study with end users. Such a user study must control factors such as device type, interface type, application type, and contexts in which tasks are performed. Ryan and Gonsalves [98] conducted a field study and found that location can significantly improve the usability of a mobile application. But how choices along key location dimensions [86] such as abstraction (e.g., position and place) and perspective (e.g., subjective, objective, and inter-subjective) affect usability remains to be studied. The results from our qualitative evaluation of usability can provide valuable guidelines in specifying hypotheses for a usability study involving real users.

3.6.3 Requirements Engineering and Formal Verification of Place-Aware Applications

Two important directions for place-aware application development research that we didn't address in this paper are requirements engineering and formal verification. As Salifu et al. [99] describe, a challenge with engineering place-aware applications is that the *monitoring* (changes in place) and *switching* (application behavior) requirements of such applications are rarely made explicit. Yet, modeling and analyzing location variability [2] during requirements phase is valuable in that inconsistencies and conflicts in location-based requirements can be detected early. Xipho [87] is our effort toward systematically capturing the contextual requirements of an application and incorporating a middleware for providing runtime support. However, we understand that specifying a complete set of place-based requirements during design is difficult since the places of interest to a user are often unknown a priori and are subject to change. In such circumstances, automated discovery, at run time, of fault *patterns* [100] could be a viable option. We will explore the possibility of incorporating such options in the Platys middleware in the future. Crowdsourcing is a promising avenue for acquiring requirements of context-aware applications. In a recent effort, we developed a method to exploit the crowd [81] for acquiring requirements and evaluated the method on acquiring smart home requirements.

3.7 Conclusions

Intelligent location-aware applications are being widely adopted. Yet, these applications are often developed in an ad hoc manner and yield apparently suboptimal user experiences. Platys seeks to address this problem and establishes through empirical evidence, for the first time, the benefits of place-aware application development.

Platys introduces place, a high-level abstraction of location that contrasts with position understood as geospatial coordinates. Employing location at the granularity of place can enable intelligent applications as well as enhance the usability and privacy-preserving aspects of

a location-aware application. Platys middleware, the crux of our framework, provides the necessary communication channels between the users of a location-aware mobile application and its developers, and the architectural support necessary for representing and reasoning about places.

The results of our empirical evaluation indicate that developers employing the Platys middleware spend significantly less time and effort than those not employing the middleware for representing and acquiring location, and enhancing the usability and privacy aspects of the application. Although developers employing the Platys middleware expend additional time and effort for acquiring the necessary background knowledge about the Platys middleware, the middleware pays off in other aspects of location-aware application development. Moreover, preparation is a one-time cost: a developer who employs Platys to develop several location-aware applications can save significant time and effort over the course of multiple applications. Our evaluation of the applications produced in the developer study indicate that location-aware applications produced using Platys are potentially (1) more usable and privacy-preserving from an end-user's perspective, and (2) easier to comprehend from a developer's perspective.

Acknowledgments

Thanks to Danny Weyns and anonymous reviewers for helpful comments.

Chapter 4

Platys Reasoner: Active and Semi-Supervised Place Learning

Now we describe how our middleware recognizes places of interest to a user. The Platys tagging subsystem collects place labels from an end-user and the event and action logger collects raw data from sensors. Then, the task of the Platys Reasoner is to recognize places (corresponding to a user's labels) from raw sensor readings.

This task can be addressed via a traditional machine learning paradigm. Specifically, *unsupervised learning* techniques seek to learn patterns in the sensor data, not requiring place labels. Typically, such approaches learn what we call as *staypoints*—sets of positions within a certain *radius* or those where a user stays for a certain *duration* [41, 78, 137]. Although staypoint-based approaches do not require labeling, they have the following shortcomings. One, staypoints do not capture subjective nuances in how users perceive places since: (a) no fixed values for *radius* and *duration* yield desired places for all users, and (b) staypoints exclude interesting possibilities in terms of spatially overlapping and dispersed places. Two, a staypoint does not carry an inherent meaning. A user may eventually need a symbolic name (hence labeling) to distinguish staypoints [69]. Three, staypoint-based approaches, often, require frequent sensor readings to find patterns in the unlabeled data. For example, [4] and [138] scan GPS every second and

minute, respectively. However, sensing consumes battery power—a limited resource on mobile devices.

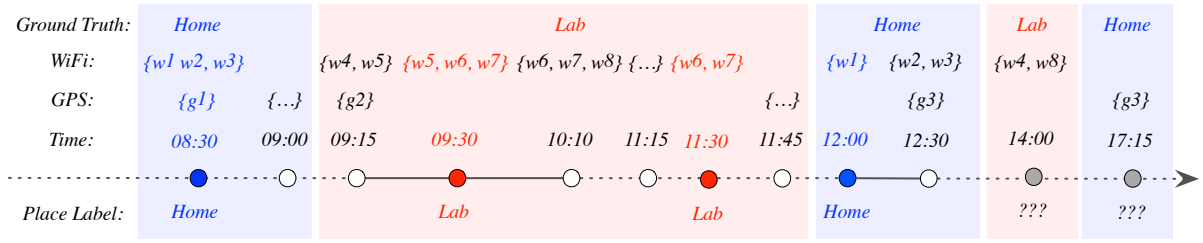
Alternatively, *supervised learning* techniques exploit user-provided place labels. A traditional classifier such as logistic regression or support vector machine (SVM) [44] can be learned from sensor data treating place labels as class labels. Typically, training a classifier requires several training instances per class to produce good classification accuracy. However, acquiring training instances is challenging because place labeling requires user effort. Requiring each user to label each place of interest several times is not practically viable. Additionally, sensor readings are likely to be (a) intermittent (all sensor readings may not be available when a user labels a place, e.g., GPS reception is limited indoors), and (b) infrequent (since sensing frequently can drain battery). Thus, many training instances constructed from sensor data are likely to be sparse (missing feature values).

The Platys Reasoner seeks to address the shortcomings of the above approaches. Specifically, it combines two machine learning paradigms: (1) *active learning* [105] to reduce the labeling effort, and (2) *semi-supervised learning* [139] to efficiently deal with intermittent and infrequent sensor data. Next, we explain these paradigms and how Platys Reasoner employs them for place recognition.

4.1 Intuition: Active and Semi-supervised Learning

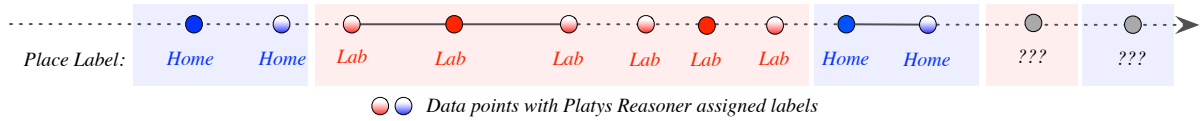
As an example, Alex is a Platys user. Figure 4.1a captures Alex's routine on a typical day: Alex is at *home* in the morning, during lunch, and evening; he works from his *lab* during a morning session and an afternoon session. As shown, Alex has labeled the two places, twice each, and there are sporadic sensor readings throughout. For simplicity (and without loss of generality), we consider only two sources of sensor readings. Let $G(t)$ and $W(t)$, and $PL(t)$, respectively be the GPS reading, WiFi scan result, and place label at time t. However, not all sensor readings may be available at each time; e.g., only WiFi scan result is available at 9:30 (GPS reception being poor indoors).

Now, we consider the questions: Given historical data (labels and sensor readings), and $W(14{:}00) = \{w^4, w^8\}$, what is Alex's place at 14:00? Similarly, given historical data and $G(17{:}15) = \{g^3\}$, what is his place at 17:15?



(a) Streams of sensor data (time not to scale). Only two sources are considered for brevity.

(b) Active learning intelligently prompts a user to assign place labels.



(c) Semi-supervised learning exploits place labels and latent structure in the unlabeled data.

Figure 4.1: An illustration of the place recognition problem and intuitions behind Platys Reasoner's techniques: active and semi-supervised learning

In general, an unsupervised approach cannot answer the above question because it does not employ place labels. In contrast, a traditional supervised approach may not be accurate given few and incomplete training instances. For example, a traditional classifier does not predict places any better than random guessing for Alex. The reason is that the instances to be predicted, i.e., $W(14{:}00) = \{w^4, w^8\}$, and $G(17{:}15) = \{g^3\}$ have nothing in common with the training instances.

The Platys Reasoner employs a classifier, albeit with additional steps in learning to address the challenges of traditional classification. Platys Reasoner's additional steps are motivated by the following intuitions.

1. Can we employ fewer training instances than traditionally required to train a classifier to achieve a desired accuracy? Yes, if we control what those training instances are (same number, though). For example, given $PL(8{:}30) = \textit{home}$ and $W(8{:}30) = \{w^1, w^2, w^3\}$, it is not useful to label $PL(12{:}00)$ as *home*, when $W(12{:}30) = \{w^1\}$. Instead, it would be better if we asked Alex to label at 12:30 as shown in Figure 4.1b. Then, we could correctly predict $PL(17{:}15)$ as *home*, unlike a traditionally trained classifier which could only guess randomly with original labels.

2. Can we exploit both labeled and unlabeled instances for training to achieve a better classification accuracy than training with labeled instances alone? Yes, if we exploit the hidden structure in the unlabeled data. For example, given that $PL(9{:}30) = \textit{lab}$ and $W(9{:}30) = \{w^5, w^6, w^7\}$, we notice that $W(9{:}15) = \{w^4, w^5\}$ and $W(10{:}10) = \{w^6, w^7, w^8\}$ overlap with $W(9{:}30)$. From this, we can assign $PL(9{:}15) = \textit{lab}$ and $PL(10{:}10) = \textit{lab}$ as shown in Figure 4.1c, and then train a classifier. Such a classifier would predict $PL(14{:}00)$ as *lab* correctly, enhancing the classification accuracy.

4.2 Platys Reasoner

Platys Reasoner incorporates the above intuitions by employing active and semi-supervised learning techniques, as shown in Figure 4.2. The reasoner operates in *training* and *prediction* modes. In the training mode, Platys starts from sporadic streams of sensor data. The active learner chooses a few instances from the pool of unlabeled data and asks a user to label them. The active learner chooses only from recently sensed data so that the user would remember what labels to use. The semi-supervised learning module picks up from where the active learner leaves off—with a few labeled instances and many unlabeled instances. The semi-supervised learner exploits the structure in unlabeled data and assigns place labels to several previously unlabeled instances. Finally, the reasoner trains a classifier from all labeled instances (user assigned and inferred). Once the place classifier is trained, given an unlabeled sensor reading,

Platys Reasoner predicts the user's place at the time of the reading.

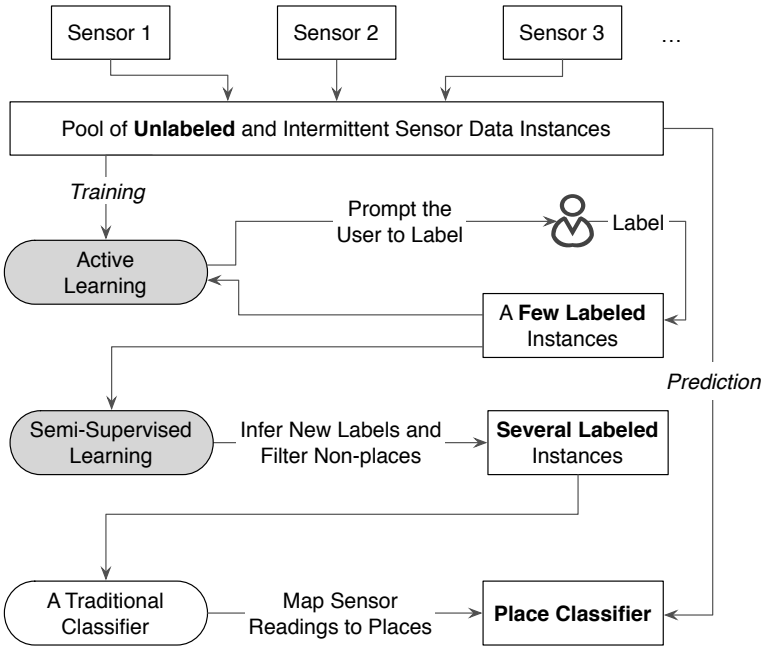


Figure 4.2: Platys Reasoner learns a place classifier from unlabeled sensor data.

Active Learning

Given a pool of unlabeled instances (from recent past) and all labeled instances (historical), our objective is to choose an instance which, if labeled, would be most beneficial in improving our classifier. Platys Reasoner adapts a technique called *uncertainty sampling* [105]:

1. *Choose the latest* unlabeled instance, if there is no labeled instance. Otherwise, perform the following steps.
2. *Train a classifier* from labeled instances alone.
3. *Predict a place label* for each unlabeled instance.
4. *Find the classifier's confidence* for each prediction.

5. *Prompt the user* to label the place for an instance predicted with least confidence.

Any classifier can be employed in the above as long as the confidence of predictions can be measured. We employ logistic regression and SVM in our analyses (Section 3.4). For logistic regression, the *probability* with which an instance is predicted as belonging to a class indicates the confidence. For SVM, the *decision value*, i.e., the distance of the instance being predicted from the separating hyperplane of the trained model, is an indicator of confidence [121].

Active learning is a continual process. As long as there are unlabeled instances that the active learner is uncertain about, it asks the user to label them. The process is robust and uses whatever information it has—a user may ignore a labeling request or proactively label a place.

Semi-Supervised Learning

The objective of semi-supervised learning is to exploit unlabeled sensor readings, given a few readings with place labels. It is effective since sensor readings tend to form well-separated clusters [24]. We employ this intuition in a semi-supervised technique called *self training* [139]. Complementary to the active learner, which asks the user when in doubt, the semi-supervised learner teaches itself from its own confident predictions. The technique operates as follows.

1. *Train a classifier* from sensor readings with user-assigned place labels.

2. *Predict a place label* for each unlabeled sensor reading via the above classifier.

3. *Retrain a classifier* from both original and newly inferred labels.

A potential problem is that the above approach assigns a place label to each sensor reading whereas some sensor readings may belong to none of the labeled places. Such readings correspond to uninteresting or novel places. Thus, we seek to filter such "noisy" sensor readings before training the final classifier. A simple strategy to filter out noisy sensor readings is to not assign a place label to an instance if the prediction confidence is below a threshold (in the second step above). However, how do we find an optimal threshold? Again, manually fixing a

threshold across all places (similar to fixing staypoints' radius or duration) is not desirable—characteristics of different places may vary. Instead, we eliminate noisy readings by iteratively clustering sensor readings as follows.

1. *Find the mean similarity* between inferred instances and original (user-assigned) instances of each place (we employ similarity metrics described in Appendix 4.2.3).

2. *Eliminate a sensor reading* from a place if the reading's similarity to original instances is less than the mean similarity for the corresponding place.

3. *Repeat the above steps* until the difference in the number of instances eliminated in two consecutive iterations is negligible.

The result of the above process is a tightly-knit clusters of sensor readings such that at least one instance in each cluster has a place label. Now, we assign the same label to all instances in a cluster and train the final place classifier.

Platys Reasoner seeks to recognize places of interest to a user from intermittent sensor readings and user-provided place labels. In this section, we formulate the place recognition problem and describe the techniques employed by Platys Reasoner to address the problem. We also provide the pseudocode for our algorithms.

4.2.1 Problem Formulation

We formulate place recognition as a machine learning problem. Further, we assume that the following sensor readings are available. These are the same sensors we employed in our user study (Section 3.4).

GPS scan results $G = \{g_1, \ldots, g_{|G|}\}$, where each g_i is a latitude-longitude pair.

WiFi scan results $W = \{w_1, \ldots, w_{|W|}\}$, where each $w_i = \{w_i^1, \ldots, w_i^{|w_i|}\}$ is a set of access points (APs) found in a scan and each w_i^j contains a MAC identifier and a received signal strength indicator (RSSI).

Bluetooth scan results $B = \{b_1, \ldots, b_{|B|}\}$, where each b_i is a set of Bluetooth devices (BtDs) found in a scan and each b_i^j contains a MAC identifier and an RSSI.

Google Places $GP = \{gp_1, \ldots, gp_{|G|}\}$, where each $gp_i = \{gp_i^1, \ldots, gp_i^{|gp_i|}\}$ is a set of Google places corresponding to a $g_i \in G$ retrieved from a web service [34] and each gp_i^j contains the name a point of interest (POI) and its distance to g_i.

Place labels $PL = \{pl_1, \ldots, pl_{|PL|}\}$, where each pl_i is a user-assigned place label.

Further, each data item above is timestamped. Let $T = \{t_1, \ldots, t_{max}\}$ be the ordered set ($t_i \leq t_{i+1}$) of all timestamps such that at least one data item is associated with each t_i. Further, let $G(t_i)$ be the GPS scan results at t_i (which can be null if GPS reading is not available at t_i). Assume similar definitions for $W(t_i)$, $B(t_i)$, $GP(t_i)$, and $PL(t_i)$. Further, let $I(t_i) = PL(t_i) \cup G(t_i) \cup W(t_i) \cup B(t_i) \cup GP(t_i)$ be the set of all sensor readings at t_i. Now, the place-recognition problem is as follows.

Given G, W, B, GP, P, and $t > t_{max}$: $I(t) \neq \emptyset$ and $PL(t) = \mathsf{null}$, what is $PL(t)$?

4.2.2 Solution Overview

Platys Reasoner addresses the place recognition problem via a classification technique. In order to do so, the reasoner prepares data as shown in Algorithm 1. In general, Platys Reasoner can employ any classifier in this algorithm (as long as the confidence of predictions can be inferred). We demonstrate this idea via a *1-Nearest Neighbor* (*1NN*) classifier because of its simplicity [118]. Training a *1NN* classifier is trivial—each labeled instance is representative of the corresponding class. Predicting from a *1NN* classifier involves finding a class whose instances are most similar to the unlabeled instance (being predicted), when compared to instances of other classes. In order to do so, we employ the similarity measures defined next.

Algorithm 1 Place classifier: Trains a classifier from labeled instances.

Require: G, W, B, GP, PL, t

$I \leftarrow \emptyset$ ▷ Training instances
for all $p \in P$ **do**
  $I_i^{pl} \leftarrow pl$ ▷ Class label
  $t_i \leftarrow PL'(pl)$
  ADDFEATURES$(I_i, G(t_i), W(t_i), B(t_i), GP(t_i))$
end for
$P_{model} \leftarrow$ TRAIN(I) ▷ Any generic classifier
ADDFEATURES$(I_{test}, G(t), W(t), B(t), GP(t))$
$pl_t \leftarrow$ PREDICT(P_{model}, I_{test})
return pl_t ▷ Place at time t

function ADDFEATURES(i, g, w, b, gp)
  $i^g \leftarrow g$ ▷ GPS features
  $i^w \leftarrow w$ ▷ WiFi features
  $i^b \leftarrow b$ ▷ Bluetooth features
  $i^{gp} \leftarrow gp$ ▷ Google Place features
end function

4.2.3 Similarity Measures

In our setting, an instance (whether labeled or unlabeled) consists of GPS, WiFi, Bluetooth, and POI features. However, some feature values may be null due to the intermittent nature of data. Given two instances, our objective is to return a value in $[0, 1]$ indicating the extent to which the two instances are similar (1 meaning most similar).

We define the similarity between (1) features corresponding to different sensors (e.g., WiFi and GPS), and (2) a feature value of *null* and anything else to be 0 since there is no meaningful comparison in these cases. We describe computation of similarity between individual features of the two instances as follows.

1. The *GPS similarity* of two instances with GPS features I_1^g and I_2^g is

$$sim(I_1^g, I_2^g) = \frac{1}{1 + d(I_1^g, I_1^g)}, \tag{4.1}$$

where $d(I_1^g, I_1^g)$ is the Euclidean distance (in km) between the two coordinates.

2. The *WiFi similarity* of two instances with WiFi readings I_1^w and I_2^w is the cosine similarity between the normalized RSSI values ($[0, 1]$) of the corresponding access points (APs). If an instance contains an AP but the other doesn't, the missing AP is added to the latter with its RSSI treated as 0.

$$sim(I_1^w, I_2^w) = \frac{I_1^w \odot I_2^w}{\|I_1^w\| \times \|I_1^w\|}, \tag{4.2}$$

where $\odot$ is the dot product operator and $\|I^w\|$ is the length of the vector I^w.

3. The *Bluetooth similarity* between two instances with Bluetooth features $sim(I_1^b, I_2^b)$ is calculated similarly to that between WiFi features, as described above.

4. The *POI similarity* between two instances with Google places I_1^{gp} and I_2^{gp} depends on the frequency of the overlapping POIs. We adapt [68] to measure the similarity as follows.

$$sim(I_1^{gp}, I_2^{gp}) = \frac{2 \times IC(I_1^{gp} \cap I_2^{gp})}{IC(I_1^{gp}) + IC(I_2^{gp})}, \tag{4.3}$$

where $IC(I^{gp}) = -\sum_{poi \in gp} \log Prob(poi)$. $Prob(poi) = \frac{n_{poi}}{\sum_{p \in gp} n_p}$ is the probability of visiting a POI, where n_{poi} is the number of occurrences of a POI. Our intuition here is that matching a rarer POI (e.g., Lake Johnson Nature Park) is more valuable than matching a more frequent POI (e.g., Raleigh).

5. Finally, the *overall similarity* between two instances I_1 and I_2 as the maximum similarity based on any of the above features.

$$sim(I_1, I_2) = \max_{f \in \{g,w,b,gp\}} sim(I_1^f, I_2^f). \tag{4.4}$$

We chose the above measures intuitively. However, the *1NN* classifier or Platys Reasoner is not tied to these specific measures. Next, we describe the techniques Platys Reasoner employs for place place recognition on top a traditional classifier.

Algorithm 2 Uncertainty sampling: Choose an unlabeled instance for labeling.

Require: G, W, B, GP, PL, T ▷ Few labels or none
Require: $sim()$ ▷ Similarity function
$L, U \leftarrow \emptyset$ ▷ Labeled and unlabeled instances
BUILDDATASET$(L, U, G, W, B, GP, PL, T)$
if $L \neq \emptyset$ **then**
  for all $u \in U$ **do**
    $u^{sim} \leftarrow \max_{l \in L} sim(l, u)$
  end for
  $u_{uncertain} \leftarrow \min_{u \in U} u^{sim}$
else
  $u_{uncertain} \leftarrow$ Remove first instance from U
end if
return $u_{uncertain}$ ▷ An instance to label

function BUILDDATASET$(L, U, G, W, B, GP, PL, T)$
  $i, j \leftarrow 0$
  for all $t \in T$ **do**
    if $PL(t) \neq \emptyset$ **then**
      $L_i^{pl} \leftarrow PL(t)$
      addFeatures$(L_{i++}, G(t), W(t), B(t), GP(t))$
    else
      addFeatures$(U_{j++}, G(t), W(t), B(t), GP(t))$
    end if
  end for
end function

4.2.4 Active Learning

Platys Reasoner employs an active learning technique called *uncertainty sampling* [105]. This technique, first, builds a model of places from the given labeled instances. Given a model of places and a pool of unlabeled instances, the active learner asks the user to label an instance for which the learner is least confident (among all unlabeled instances) of predicting a place. Algorithm 2 illustrates the uncertainty sampling technique. First, it builds a dataset consisting of labeled and unlabeled instances. Note that if there are no labeled instances, the algorithm arbitrarily selects an instance and asks the user to label it. Next, the algorithm employs the similarity function we defined earlier to predict labels and the similarity value as the confidence.

Algorithm 3 Self training: Infer labels for unlabeled instances.

Require: G, W, B, GP, PL, T ▷ Few labels
Require: $sim()$ ▷ Similarity function
1: $L, U \leftarrow \emptyset$ ▷ Labeled and unlabeled instances
2: BUILDDATASET$(L, U, G, W, B, GP, PL, T)$
3: **while** $U \neq \emptyset$ **do**
4: $u \leftarrow$ Remove first instance from U
5: $l_{nearest} \leftarrow \max_{l \in L} sim(u, l)$
6: $u^{pl} \leftarrow l^{pl}_{nearest}$ ▷ 1-nearest neighbor
7: $L_{i++} \leftarrow u$
8: **end while**
9: **return** L ▷ All labeled instances

4.2.5 Semi-supervised Learning

Platys Reasoner employs a semi-supervised technique called *self training* [139] to exploit both labeled and unlabeled instances. In contrast to the active learner which asks the user to teach, the semi-supervised learner teaches itself from its own *confident* predictions. Algorithm 3 illustrates self training. Similar to the active learning algorithm, we first build labeled and unlabeled instances. Next, we assign an unlabeled instance to a class based on the similarity of the instance to the class' instances.

Algorithm 4 Iterative clustering: Filter instances not belonging to any labeled place.

Require: PL, I, L ▷ All labeled instances
1: **for all** $pl \in PL$ **do**
2: $I_{pl} \leftarrow I(PL'(pl)$ ▷ Originally labeled pl
3: $L_{pl} \leftarrow L(PL'(pl))$ ▷ Assigned to pl
4: $\epsilon, \epsilon' \leftarrow 0.5$ ▷ Similarity
5: $\delta = 0.01$ ▷ Convergence threshold
6: **repeat**
7: $\epsilon \leftarrow avg_{l \in L_{pl}, i \in I_{pl}} sim(l, i)$
8: **for all** $l \in L_{pl}$ **do**
9: **if** $avg_{i \in I_{pl}} sim(l, i) < \epsilon$ **then**
10: $remove(l, L_{pl})$ ▷ *Filter out*
11: **end if**
12: $\epsilon' \leftarrow avg_{l \in L_{pl}, i \in I_{pl}} sim(l, i)$
13: **end for**
14: **until** $|\epsilon - \epsilon'| > \delta$ ▷ until convergence
15: **end for**
16: **return** L ▷ Several labeled instances

The self-training algorithm assigns each unlabeled instance a place label. However, a user may not have labeled all places he visits. Also, not all (sets of) positions might be of interest to a user. Thus, assigning a place label ($p \in P$) to each unlabeled instance can mislead the learning algorithm. In order to address this problem, we consider an iterative clustering algorithm that filters out sensor readings that belong to none of the user's labeled places. Algorithm 4 illustrates the iterative clustering technique. Our intuition is to find an appropriate *similarity boundary* for each place such that the boundary groups sufficiently similar data instances as belonging to the corresponding place. Then, we filter out instances outside the boundary. Our approach begins with a fairly large similarity boundary (with similarity, $\epsilon = 0.5$); iteratively clusters a set of instances; and reduces the similarity boundary (i.e., increases ϵ) based on the mean similarity (ϵ') of the currently clustered instances until the boundary converges.

4.2.6 Platys Social: Recognizing Ego-Centric Social Circles

The place recognition pipeline described above is generic in the sense that it can incorporate multiple sensors. Sensors available on a typical mobile device today provide clues about a

place's spatial attributes (e.g., via ambience sensors [5]) and the activity component (e.g., via accelerometer [63]). However, how do we recognize the third component of our place metamodel—social circles?

Traditionally, *community detection* [30] from online social networks (OSNs) is used for recognizing social circles. However, such an approach is not suitable for our setting because of the following reasons. First, community detection from an OSN presupposes that the global network structure is known. However, such information is not available to end-users. Second, communities detected from an OSN are typically much coarser than social circles in real life, e.g., all of a users friends from *college* are likely to be in one OSN community (based on mutual acquaintanceship), whereas the user may perceive multiple social circles within the *college* community.

Platys Social [85] is our approach for recognizing ego-centric social circles of a user. The approach is based on the intuition that a user is likely to perceive a set of *contacts* (other users) as a social circle if the user meets those contacts together, regularly. We employ Bluetooth technology to identify spatial proximity between users because of its short range and widespread availability on mobile devices. However, many contacts of a user are not likely to be Bluetooth discoverable. Thus, social circles so discovered are likely to be sparse. We address this problem by incorporating information from the user's real life interactions as follows.

1. *Construct a contact co-occurrence graph* based on the spatial proximity between the contacts observed over time. Each contact of a user is a node in the graph. There is an edge between two contacts if the user meets the two contacts together. The weight of an edge is proportional to the frequency of meetings.

2. *Extend the co-occurrence graph* to incorporate information from interactions via emails, phone calls, and instant messages. That is, add an edge between two contacts if the user included both contacts in an interaction. If an edge already exists, update the weight according to frequency of co-occurrence.

3. *Detect communities* from the co-occurrence graph and treat each community as a social circle. We employ the weighted clique percolation method [89] to detect overlapping communities since social circles are likely to overlap.

Although Platys Social employs community detection, a difference from traditional approaches is that it detects communities in a graph constructed from real life proximity and interactions. Further, Platys Social incorporates only the local information about a user available to the Platys middleware.

4.3 End-User Study

We evaluated Platys Reasoner via a user study. We analyzed the accuracy with which the reasoner recognizes places of interest to a user and its efficiency in doing so.

4.3.1 Data Acquisition

No available datasets were adequate for our evaluation. We created our own dataset based on real traces collected from 10 users. Each user carried an Android phone installed with Platys middleware as his or her primary phone for three to 10 weeks. The middleware collected a user's place labels and recorded GPS, WiFi, and Bluetooth readings. The study was approved by our university's Institutional Review Board (IRB).

Platys Reasoner's objective is to exploit infrequent and intermittent data. The middleware invoked sensors only when a user labeled a place (a few times a day). However, the middleware, a background service, always listened to the sensors. Thus, the middleware received data from a sensor even when other applications invoked that sensor.

In real use, Platys Reasoner learns and predicts places continually. However, we disabled the reasoner's learning modules during data acquisition, enabling users to label places without any bias and to avoid the possibility that if the reasoner were to begin predicting a place accurately, the user might stop labeling that place, thereby providing us insufficient ground

truth for evaluation. The middleware reminded users to label their current place at random intervals. Thus, we captured how a user naturally labels places, which we use as a baseline to evaluate active learning.

Table 4.1 summarizes the data we acquired. Our dataset contains a variety of users (one faculty member, one postdoc, and eight graduate students from two departments; seven male and three female), differing in their mode of transportation (drive or walk), mobility across states and countries, and frequencies of sensor data collection.

Table 4.1: Summary of the data acquired in user study.

User	**Study days**	**All labels**	**Unique labels**	**GPS scans/day**	**WiFi scans/day**	**Bluetooth scans/day**
A	70	173	18	19	140	73
B	38	63	9	11	79	87
C	68	82	14	7	19	21
D	37	128	9	199	763	0
E	48	32	4	10	129	44
F	24	40	3	22	50	0
G	70	340	11	9	323	65
H	63	38	6	113	408	37
I	21	36	9	5	208	45
J	21	56	9	12	220	3
Mean	45	94	9	41	234	38

4.3.2 Evaluation Metrics

We treat place recognition as a classification problem and evaluate its performance via precision = $\frac{TP}{TP+FP}$, recall = $\frac{TP}{TP+FN}$, and F-measure = $2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$, where TP, TN, FP, and FN refer to true and false positives and negatives.

Typically, these metrics apply to a binary classification problem. However, place recognition involves multiple classes (each place is a class). Thus, we use the *one-versus-the-rest* strategy [10] in which we calculate a per-class F-measure for each place as a class, treating rest of the

places as another class. Then, we assess the overall place recognition accuracy by averaging the per-class F-measures.

4.3.3 Comparison with Two Traditional Classifiers (Supervised)

Platys Reasoner employs a traditional classifier and the benefits it offers arise due to active and semi-supervised learning enhancements. We evaluated the benefits of each enhancement on logistic regression and SVM.

Active Learning

To evaluate the claim that Platys Reasoner's active learner reduces place labeling burden, first, we temporally ordered all labeled instances corresponding to a user. Recall that our learning algorithms were disabled during data acquisition so that the order in which labels were assigned was user controlled or random if the user simply labeled when the middleware reminded the user to. Retrospectively, we want to check what would have happened had the places been labeled according to the active learner's expectations. Thus, for a given number of labels n, we trained a traditional classifier employing n labeled instances in the order that (1) the user labeled them, and (2) the active learner would have expected the user to label them. We used the rest of the labeled instances for testing.

Figure 4.3 shows a comparison of F-measure, averaged across users, of the two classifiers and their active learning versions (semi-supervised learner was not used in these comparisons). We stop at $n = 7$ since eight is the maximal number such that each user labeled at least two places eight times in our dataset (we need at least two classes to train a classifier and at least one labeled instance to test).

We found a difference in the baseline and active learning versions of the classifiers with as few as three labels. For example, at $n = 3$, an active learning version of logistic regression performs on par with the corresponding baseline at $n = 6$. This supports our claim that an active learner can significantly reduce a user's place-labeling effort.

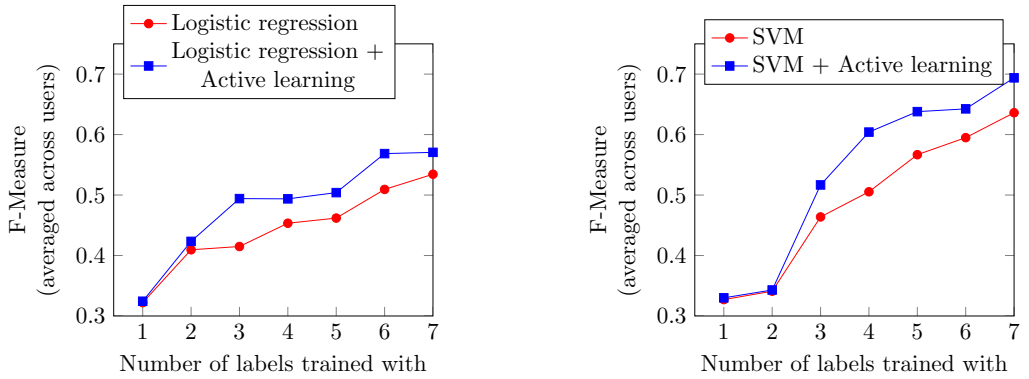


Figure 4.3: Platys Reasoner's active learner compared with two traditional classifiers.

Semi-Supervised Learning

We claim that Platys Reasoner's semi-supervised learner, which employs both labeled and unlabeled instances, recognizes places with better accuracy than a traditional classifier which employs labeled instances only. We evaluated this claim via SVM. As shown in Figure 4.4 (left), the semi-supervised SVM achieves a higher F-measure than SVM.

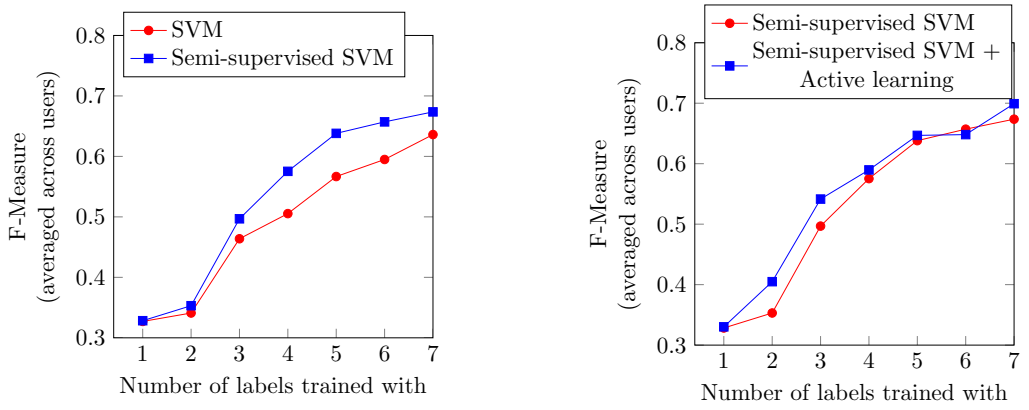


Figure 4.4: Platys Reasoner's semi-supervised and active learning compared for SVM.

Our place-recognition pipeline employs both semi-supervised and active learning techniques. Figure 4.4 (right) demonstrates the complimentary benefits of the two techniques. First, with

a few place labels ($n \leq 3$ in our dataset), active learning improves the semi-supervised SVM's F-measure noticeably. Next, with several place labels (accordingly, more sensor readings), semi-supervised SVM's F-measure is in par with its active learning version. That is, whereas active learning is valuable in the initial phases of training, semi-supervised learning can compensate for a user's non-compliance to place labeling requests in later phases of training.

4.3.4 Comparison with Two Staypoint-Based Approaches (Unsupervised)

We compared Platys Reasoner with two staypoint-based approaches [41, 137]. However, the comparison was nontrivial for three reasons. First, a staypoint-based approach requires fixed values for place radius and duration. Since the optimal values for these parameters are not obvious, we varied them from $\langle$3 minutes, 20 m$\rangle$ to $\langle$1 day, 96 km$\rangle$.

Second, a staypoint-based approach does not distinguish one staypoint from another. Thus, it can only predict whether a data instance belongs to a staypoint or not. To make a fair comparison, we implemented a variant of Platys Reasoner called *Place-or-not* that distinguished whether a data instance belongs to one of the labeled places or not (and call the version that recognizes specific places as *Which-place*).

Finally, a staypoint-based approach requires several sensor readings to perform well. Although the approach itself does not require labels, our evaluation requires labels as ground truth. Since only a few sensor readings are labeled in our dataset, we requested our users to provide additional ground truth. Six of the original 10 users did so. For each of the six users, we provided a web-based (large-screen) interface showing sensor readings and asked the user to indicate whether the user was in one of labeled places or not at the corresponding time. To assist users in recalling this information, the interface showed GPS coordinates on a map, provided other sensor readings at that time as well as the user's previous and next labeled place.

Figure 4.5 compares the F-measures for Platys Reasoner and two staypoint-based approaches. Our findings are three fold.

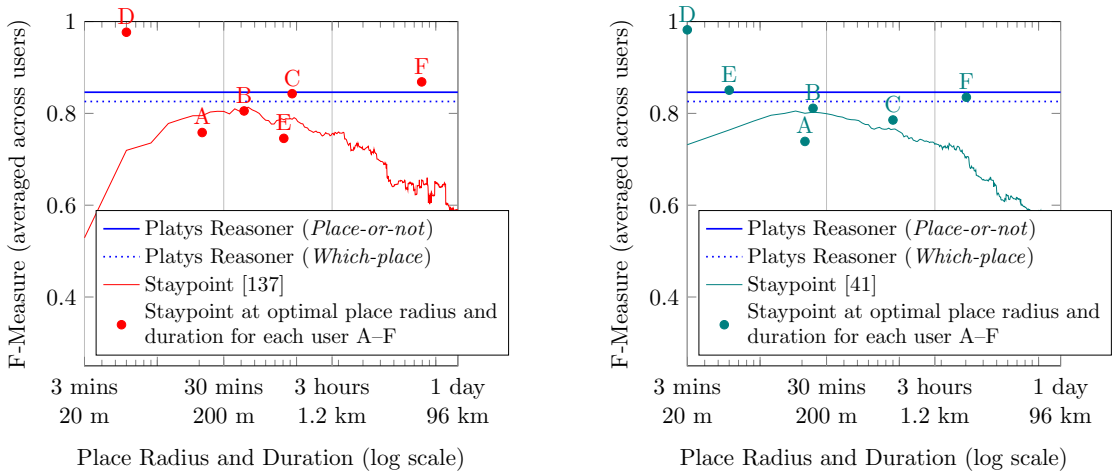


Figure 4.5: Platys Reasoner compared with two staypoint-based approaches.

(1) Place-or-not performs better than both staypoint approaches we compared with. The F-measures for Platys Reasoner, unlike those of staypoint approaches, are straight lines since since they do not depend on place radius and duration.

(2) The parameters ⟨30 minute, 200 m⟩ used by [137] are reasonable, but not optimal for all users (dots in the figure indicate individually optimal values).

(3) Place-or-not is an upper bound on Which-place. However, in most cases, the two F-measures are close. That is, once Platys identifies a user to be in one of the labeled places, in most cases it correctly identifies which place the user is in.

4.3.5 Related Work

Existing techniques that seek to recognize places are predominantly unsupervised. These approaches typically recognize staypoints, an abstraction richer than position, but cover (to varying extents) only the spatial aspect of places.

Ashbrook and Staner [4] collect GPS logs once per second if the user is moving beyond one mile per hour and apply a variant of k-means clustering to extract places. Similarly, Zhou et al. [138] learn places from one-minute frequent GPS logs, though via density-based clustering, and

obtain better accuracy than the k-means approach. NextPlace [102] models the importance of a GPS coordinate to a user as a Gaussian distribution based on the the user's length of stay at the coordinate and at the coordinates next to it. NextPlace considers, as places, only those coordinates that have an importance higher than a specified threshold. Similarly, Zheng et al. [137] and Hariharan and Toyama [41] extract staypoints via clustering (with fixed staypoint parameters $\langle$30 minute, 200 m$\rangle$) and probabilistic approaches, respectively.

Unlike the GPS-based approaches above, Kang et al. [57] learn places based on a location database of WiFi access points. They sort the locations of WiFi access points based on time; group proximate locations as a cluster; create a new cluster when a location is far away from the current one; and ignore the clusters within a short period of time. Kang et al.'s idea is quite similar to the GPS staypoint-based approaches. Vu et al. [122] apply star clustering on a co-occurrence graph of WiFi access points.

Another popular category of place-recognition approaches employ cell-towers logs. A cell-tower, similar to a WiFi access point, broadcasts its unique identifier. Cell phones can periodically scan for the identifiers of nearby cell-towers. Hightower et al. [47] extract a place by seeking a *stable scan*, which occurs when there is no new cell-tower or WiFi signal seen within a certain period of time. Their approach requires highly frequent scans (2Hz). Similarly, SensLoc [58] detects a user's entry and departure from a place based on the stability of cell-tower signals; recognizes places using cell-tower and WiFi signals; and, tracks movement paths between places using GPS. SensLoc conserves power by stopping unnecessary sensor scans when a user has no movement, as detected by an accelerometer.

Chapter 5

Platys Social: Relating Places and Social Circles

5.1 Introduction

Users today increasingly participate in online social interactions, especially media-driven interactions that may have no offline correlate. Although online interactions can be rewarding for users, they open up new challenges.

Control: how flexible can a user be in choosing with whom he interacts?

Cognitive overload: can a user prioritize interactions and information so as to reduce his cognitive burden?

Data privacy: can we support the above without storing a user's private information outside of his personal devices?

A major source of the first two of these challenges is that online relationships today exhibit a flat structure, or as Deresiewicz [18] puts it, everyone in the online world is a faux friend. By contrast, in traditional (offline) settings, users implicitly categorize their connections into multiple *social circles*, such as family, classmates, colleagues, friends from different cities, and so

on. Further, a user may have different priorities among their connections. Recognizing a user's social circles and priorities can benefit several applications.

Social network sites, supporting (i) friendship suggestions, (ii) fine-grained privacy policies, and (iii) enhanced social search by ranking paths to a target individual [40].

Email, facilitating email triage by prioritizing incoming messages.

Social virtual worlds, mapping the offline social circles of a user to his avatar in a virtual world [45].

Social networking applications increasingly support users structuring their connections ("groups" on Facebook; "circles" on Google+). Manually creating social circles and prioritizing their member connections, especially as they change over time, is tedious and time consuming [97]. Lampinen and colleagues [65] describe an extensive study that highlights the challenges in privacy for users and the inadequacy of using static groupings of connections. Grouping is not effective without prioritization of a group's members. Thus, we need automatic approaches for recognizing social circles and priorities.

Community detection [30] is a widely used approach for identifying groups of users in social networks. Informally, a *community* in a network is a set of nodes with dense edges within the set, and sparse edges to the rest of the network. However, existing social networks merely include acquaintance relationships; communities in such networks are coarser than social circles. For example, a user's college connections may all fall into one community if they have sufficiently many mutual acquaintances, despite reflecting many social circles. Further, detecting communities in a social network presupposes knowing the global network structure, which makes the approach infeasible unless users provide their private data to a third party.

We propose a novel approach called *Platys Social* that addresses the above challenges through the following characteristics.

Automatically learning and maintaining the social circles of a user and the priorities among the connections in each circle.

Exploiting contextual information and offline user interactions for learning, yielding social circles that are more meaningful than the communities based merely on acquaintance relationships.

Privacy preserving by employing only the information locally available to a user and not storing private information outside of a personal device.

Let us define two key terms.

- A user's *connection* is anyone the user recognizes and relates to in some context. A connection of a user may have multiple identifiers—offline or online—such as appearance, name, phone number, email ID, Bluetooth device address, and so on.
- A *social circle* of a user is a set of connections the user perceives as a logical group. A user's social circles are ego-centric in that they are defined from a user's perspective, not necessarily from the connections' perspectives.

5.2 Social Circles and Connection Priorities

Platys Social seeks to address the following main questions.

What is a natural basis for logically grouping a user's connections? We propose the notion of a *place-based identity*. A place, contrasted with geospatial position, is a conceptual construct with high salience for user actions and interactions [42]. A typical user visits several logical places and shares such places with others. Examples of *shared places* include home, workplaces, classrooms, friends' homes, restaurants, and so on. It is interesting that a user can identify most of his connections in conjunction with such shared places. For instance, family members can be typically identified with one's home, classmates with classrooms, coworkers with one's workplace, and so on.

How can we prioritize the connections in a social circle? We propose to do so based on the frequency of interactions. Platys Social categorizes a user's connections into two main categories as follows.

- A *strong connection* is one with whom the user interacts frequently.
- A *weak connection* is one with whom the user interacts infrequently.

5.2.1 Platys

Our informal answers above presuppose a framework for gathering user information such as the places he visits, his connections, interactions, and so on. *Platys* (`http://sites.google.com/site/platysproject/`) is an active effort in building a framework for (i) efficiently sensing the low-level information about a user such as his position, environment, and actions, (ii) learning the high-level concepts such as the places and social circles from the sensed information, and (iii) supporting intelligent applications that exploit place and social circles.

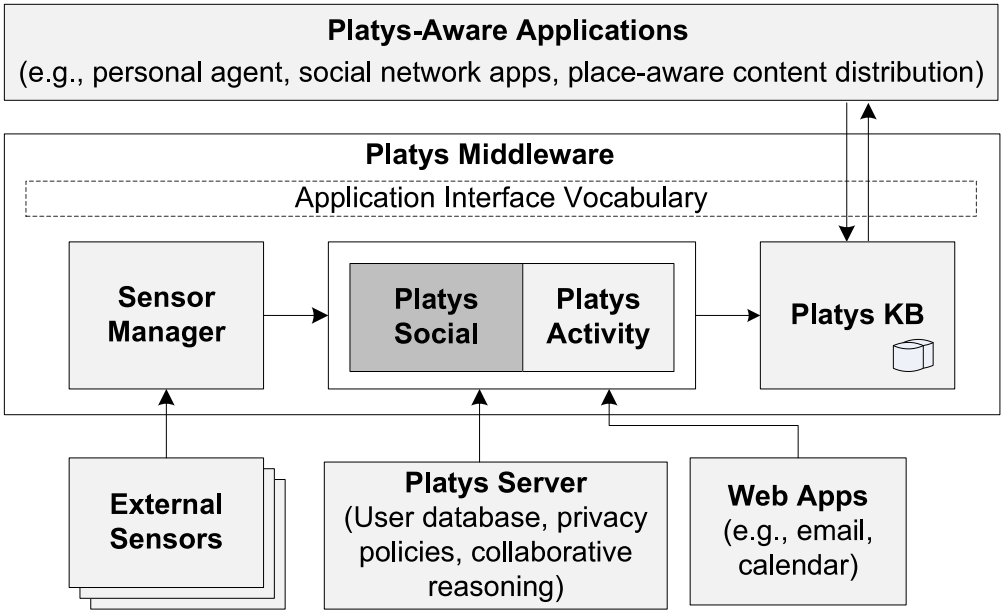


Figure 5.1: The Platys architecture, highlighting the focus of this paper.

Figure 5.1 shows the Platys architecture as consisting of three major components: sensors, middleware, and applications. In principle, all these components can be installed on a user's personal device. Smart phones are our devices of choice: they come with a variety of sensors; are almost always carried by a user; and are increasingly powerful.

5.2.2 Platys Social

Platys Social is a component of the Platys middleware. Figure 5.2 shows its structure.

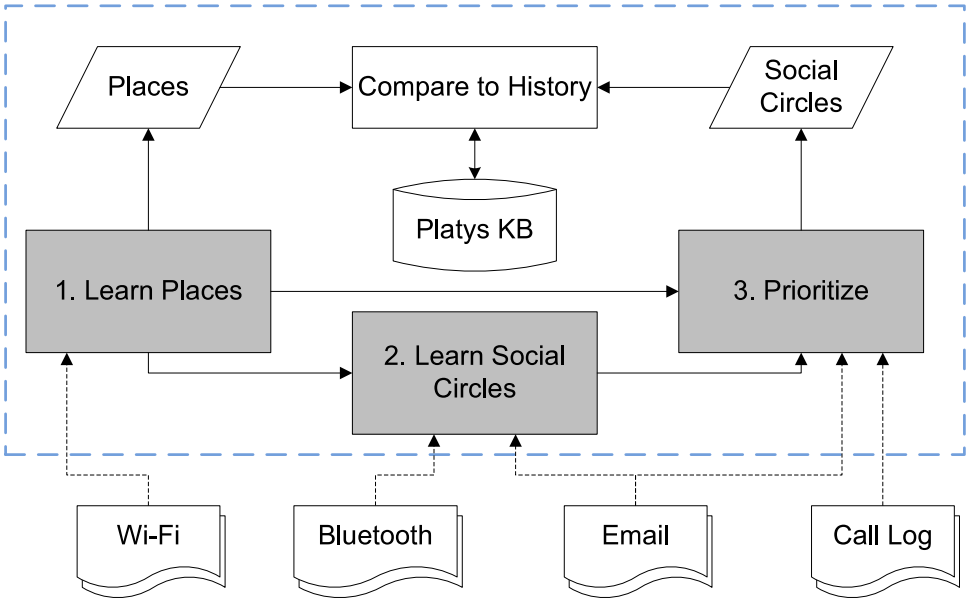


Figure 5.2: The architecture of Platys Social, highlighting its learning modules.

Place Learning

In order to group a user's connections using place-based identities, we should first identify the socially significant places a user visits. Platys Social identifies such places by exploiting the prevalence of Wi-Fi access points (APs) in modern urban environments. The sensor manager in a Platys-enabled device continually scans Wi-Fi channels. For each scan, the middleware logs

a timestamped vector of APs, where each AP is associated with a (i) unique address (*BSSID*), (ii) user-defined name (*SSID*), and (iii) Received Signal Strength Indicator (*RSSI*).

Considering each scan event in the Wi-Fi AP log as a data point, we perform cluster analysis to discover significant places. Clustering presupposes a distance measure between any two data points, which we define as the so-called cosine distance

$$cos_{\delta}(i,j) = 1 - \frac{rssi_i \odot rssi_j}{\parallel rssi_i \parallel \times \parallel rssi_j \parallel}, \tag{5.1}$$

where $rssi_i$ and $rssi_j$ are vectors of *RSSI* values for scan events i and j of lengths $\|rssi_i\|$ and $\|rssi_j\|$, respectively, and $rssi_i \odot rssi_j$ represents their dot product.

A challenge we face in clustering APs is that the number of clusters in the data (number of places a user has visited) is unknown a priori. We build a dendrogram of APs using Matlab's hierarchical clustering package (`http://www.mathworks.com/products/statistics/`). We use the distance that maximizes the silhouette coefficient (which combines cluster separation and cohesion [118]) to cut the dendrogram. Once the clusters are computed, we ignore clusters with APs of low RSSI values as noise. Each remaining cluster corresponds to a place and is associated with (i) a set of consistent APs, (ii) a cumulative RSSI value for each AP, and (iii) a set of timestamps.

Social Circle Learning

Once we identify the places a user visits, how may we identify people in those places? Bluetooth appears to be a promising technology—it has a short range and most users have mobile devices equipped with it. Thus, the Platys middleware scans for Bluetooth devices continually and records a Bluetooth device log similar to the above Wi-Fi AP log. To learn the social circle corresponding to a place, we group all Bluetooth devices found in the intervals corresponding to the timestamps associated with the place. This leads to social circles that contain (i) a set of Bluetooth devices, (ii) a cumulative RSSI value for each Bluetooth device, and (iii) a place.

The above technique relies on users to keep their Bluetooth devices in discoverable mode,

which is not a popular practice, despite Bluetooth technology being increasingly secure and energy efficient. To overcome this lack of Bluetooth data, we combine place-based grouping with email-based and call-based grouping. Our intuition is that just as we group a user's connections based on shared places, so can we group them based on co-occurrence in email threads and phone calls (such as in a conference call).

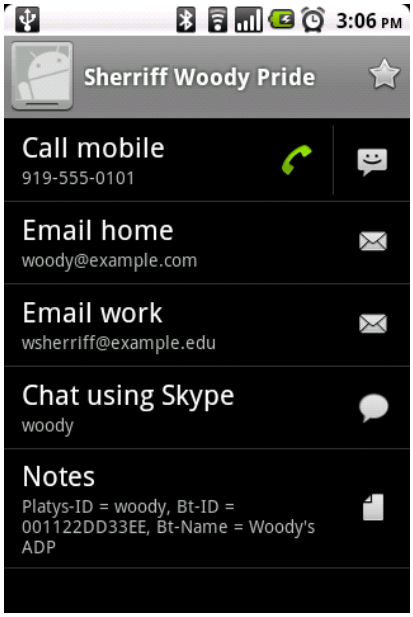


Figure 5.3: Details of a user's connection maintained by Platys.

Let $C_u = \{c_1 \dots c_n\}$ be the set of all connections of a user u. As Figure 5.3 illustrates, Platys aggregates multiple identifiers for each connection of a user in an address book. We define a weight w_{ij} for each pair of connections $c_i, c_j \in C_u$ as the weighted average of the frequency of (i) the co-occurrence of the two connections in a place, (ii) the co-occurrence of the two connections in an email thread, and (iii) the co-occurrence of the two connections in a phone call. Further, we construct a *connection co-occurrence graph*, which is an undirected and weighted graph whose vertex set is C_u, and there exists an edge between c_i and c_j if and only if $w_{ij} > 0$.

Unlike an acquaintance network, the connection co-occurrence graph is based on real interactions and contextual information. In addition, such a graph can be fully constructed using only the local information available to a user. A user's social circles can be learned by identifying communities in his connection co-occurrence graph. We apply the clique percolation method (CPM) [89] to identify the communities. An advantage of CPM is that it finds overlapping communities and the social circles of a user are likely to overlap. CPM works by identifying k-cliques in the graph and constructing communities as a union of adjacent k-cliques (two k-cliques are adjacent if they have $k-1$ nodes in common). As CPM suggests, we choose $k = 4$ and lower the weight threshold until the largest community found is twice the size of the second largest community.

Prioritizing Connections

Once the social circles are learned, we prioritize a user's connections within each social circle on the basis of the user's interactions with them. We consider the following types of interaction between the user and each of his connections.

- Face-to-face interactions estimated by the cumulative Bluetooth proximity of a connection to the user in the place corresponding to a social circle.
- Email interactions measured by the number of email exchanges between a connection and the user.
- Phone interactions measured by the number of phone calls between a connection and the user.

Accordingly, for each connection, we define an *interaction weight* as a weighted sum of face-to-face, email, and phone interactions. Platys Social designates a connection as *strong* if its interaction weight exceeds a threshold, and as *weak* otherwise. The threshold can be set by plotting interaction weights of all connections in a social circle and choosing a point that

separates a few from the many. Such a threshold reflects the intuition that social circles have a few strong connections but many weak connections.

Maintaining Social Circles

Platys Social employs an incremental approach to learning to keep a user's social circles up-to-date. Typically, each week it learns (separately for each user) places and social circles, and prioritizes connections. It then compares the learned places and social circles with a history of places and social circles, which can be used to track how the social circles evolve. An advantage of this incremental approach is that each execution of Platys Social involves amounts of data feasible for analysis on a resource-limited personal device.

5.3 User Study

We conducted a study of six users, all graduate students in their twenties and thirties, who used a Platys-enabled Android phone as their primary phone for ten weeks. The Platys middleware ran as a background service on the phone and recorded Bluetooth and Wi-Fi scans every five minutes. In addition, the middleware could access the user's email and call logs.

In order to acquire the ground truth, we asked each user to maintain a *place calendar* by updating a calendar with all socially significant places they visited each day (home, classrooms, workplaces, restaurants, and so on). Towards the end of the study, each user identified social circles corresponding to the places in their place calendar. In addition, each user prioritized connections in each social circle as strong or weak. The concepts of social circles, strong and weak connections were informally described to the users to capture their natural intuitions.

The learning process was offloaded to a server due to the lack of data analysis software for Android. There are obvious privacy concerns because the server was accessible to the researchers of this study. However, in practical deployment, the server can be thought of as hosting users' personal agents. It need not be a server of the conventional social network site that collects user information.

5.3.1 Place Learning

In order to evaluate the place learning, we compared the places learned by Platys Social and the places reported by the user in the place calendar. We define two places to be similar if the overlap between the timestamps associated with the two places is greater than a predefined threshold. The Jaccard similarity [118] between the learned and the reported places is

$$Place\ Similarity = \frac{|\ Learned\ places \cap Reported\ places\ |}{|\ Learned\ places \cup Reported\ places\ |}. \tag{5.2}$$

Figure 5.4 shows the *Place Similarity* for each user. The plot indicates that Platys Social is effective in learning places with similarity averaging nearly 85%. We further investigated the places not common between the learned and the reported, and uncovered interesting reasons for such errors: (i) sometimes, the users reported two learned places such as two shops in a mall as one place (ii) some learned places were pass-by places that the users didn't identify as significant, and (iii) some reported places with poor Wi-Fi infrastructure were discarded as noise by Platys Social.

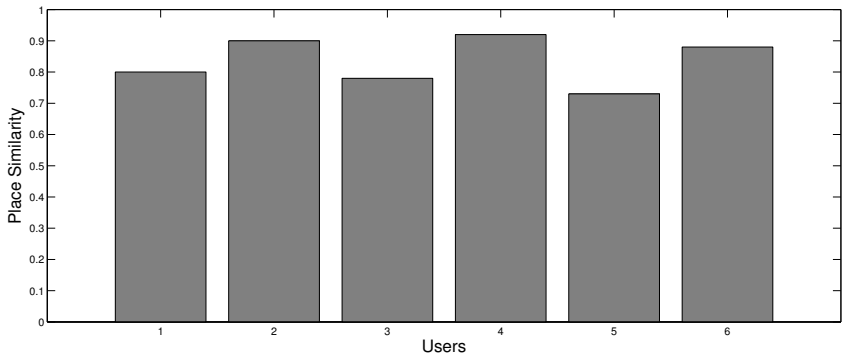


Figure 5.4: Similarity between places learned by Platys Social and manually identified by users.

5.3.2 Social Circle Learning

Similar to places, we also evaluated the similarity between the learned and reported social circles. We define *Circle Similarity* by replacing places with social circles in Equation 5.2.

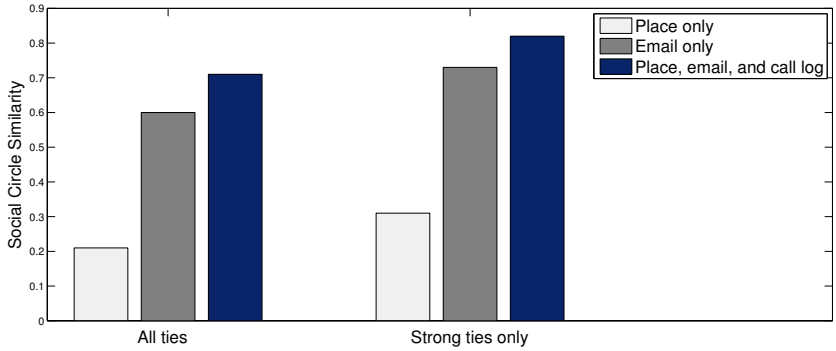


Figure 5.5: Similarity between social circles learned by Platys Social and the social circles reported by users.

Figure 5.5 compares *Circle Similarity* for different criteria, averaged across all users. The email history alone is more effective than using the place information alone. Although we claim that the place-based identities is an elegant mechanism, the reason for its relative ineffectiveness is that many users today are not Bluetooth-discoverable, and the social circles learned contained fewer users than expected. However, it is interesting that combining place and email information enhanced the effectiveness of social circle learning. Looking into the details, we found that users did not use email to interact with all of their social circles. For example, not surprisingly, some users had insignificant email interactions with their family members despite meeting them regularly, as identified from the place information. Our data didn't include any conference calls (unsurprising outside of business settings). Thus we couldn't evaluate the effectiveness of phone calls in learning social circles.

Finally, we analyzed only the strong ties learned by Platys Social and reported by the users. We found that the heuristic used by Platys Social (with place, email, and call logs) learned the

strong ties of the users more effectively than all ties. Although Platys Social successfully learned most user-reported weak ties, it learned unreported ties as well. We conjecture that such false positives correspond to *familiar strangers* [91], whom we encounter often albeit without any direct interactions (suitably extending the notion to email threads).

5.4 Directions

Platys Social opens up several avenues for future research. First, a Wi-Fi cluster does not quite capture a logical place that Platys envisions. For example, a user may perceive two *seminar halls* to be the same place, despite the two being different rooms. On the contrary, a user may view a coffee shop as two places, both a *caffeine fix* and a *meeting place*. We propose that recognizing user activities can serve to enhance the notion of place. For example, what makes two seminar halls in distant corners of a campus the same place is that similar activities take place in both. A key challenge in activity recognition is to bring together information from various sources such as sensors, browsing data, application usage, and so on [107]. In addition, understanding a user's mobility patterns [96] can provide useful hints for activity and place recognition. For example, a user's mobility pattern in a *theme park* might be quite different from that in a *poster session*.

Second, the *strength* of a connection (tie) classically incorporates the amount of time spent interacting, emotional intensity, intimacy, and reciprocal services that characterize the tie [35]. However, Platys Social captures only the frequency of interactions. It remains to be seen if frequency is an effective surrogate for the other factors and what easy-to-compute attribute may supplement frequency. Frequency alone proves inadequate in some settings. For example, a next-door neighbor may be incorrectly prioritized as a strong connection because of frequent face-to-face interactions. Platys Social can potentially benefit from technologies such as the Sociometer [17], which attempt to model face-to-face interactions.

Third, Platys Social requires manual effort to aggregate multiple identifiers of a user's connections. Performing this task automatically and in a privacy preserving manner is a significant

challenge and is essential for wider adoption.

Fourth, Platys Social exploits only the relationship from places to social circles. The implications of the relationship between social circles and places remains to be studied. For example, knowing that two of a user's significant places have social circles with same members is an indication that the two places might be logically the same place.

Fifth, the ten-week duration of our study precludes an effective examination of the changes to users' social circles. Future enhancements to Platys Social and studies over longer durations would help us address the above challenges.

5.5 Conclusions

A user's mobile and social contexts influence each other. I am interested in studying the influence in both the directions. (1) In Platys Social, we studied how the locations visited by a user (mobile context) can predict his social relationships and their strengths. (2) Conversely, in Percimo [133], we studied how the user's communities (social context) can predict the locations he visits.

Understanding and exploiting the interplay between mobile and social contexts can yield important applications domains such as recommender systems, urban planning, and health monitoring. However, existing works in this area typically study only on one aspect of context, i.e., location. I see immense value in understanding how richer contextual attributes such as user activities and sentiment [134] influence social relationships and norms.

The ideas demonstrated in Platys Social could naturally be combined with a variety of software applications that involve interaction among people: these include not only email, chat, and social networking, but also ad hoc business processes. Platys Social can enhance user experience by helping structure and prioritize not just information flow but a user's actions generally in a manner that is socially salient for that user. Further, because Platys Social takes an ego-centric stance, it is naturally privacy preserving. When implemented on a user's personal device, it could avoid many of the risks associated with sharing information via a third party.

Chapter 6

Engineering Privacy-Preserving Personal Agents

A social network service (SNS) enables users to maintain social relationships via online interactions. SNS users share information with each other as they interact. Often, the information shared on an SNS involves several users (e.g., a photo showing a group of people). Many SNSs enable users to connect the information they upload to other users so that the connected users can be notified of the uploaded information. Since the information shared varies depending on the SNS, these connections can take different forms, e.g., tags on a photo uploaded to Instagram or mentions in a tweet. Suppose Alice uploads a photo from last weekend's party where she and her friend Bob appear together, and tags Bob in the picture. When these connections are created, the other users are linked to the uploaded information. Usually, a connection implies that the profile of the user can be accessed from the information or some personal information is shown in conjunction with the uploaded data. Although connections between information and users are widely employed by SNSs, they can pose a privacy threat. For example, Bob may find that the photo Alice uploaded is sensitive. However, since Bob has no control over uploading that photo, Alice's action can threaten Bob's privacy. We identify a situation such as this as a *multiuser privacy scenario* (*multiuser scenario*, for brevity) [28].

Currently, SNSs do not provide mechanisms to handle multiuser scenarios [29]. Thus, a user who did not upload a piece of information concerning him must deal with the privacy settings chosen by the uploader; at best, the user can remove the connection that links him to the shared information, but the information itself remains nevertheless. An ideal solution in a multiuser scenario is to respect each user's privacy. However, often such a solution may not be viable since the preferences of the users involved may conflict. For example, suppose Alice would like to share a photo in which Bob and she appear with her friend Charlie. However, Bob would like to share it only with his common friends and he does not know Charlie. Here, no solution completely respects both Alice and Bob's preferences.

Several researchers, e.g., [8, 65, 116, 131], have identified decision-support systems that help users resolve multiuser privacy conflicts as one of the biggest gaps in privacy management in social media. The main challenge that decision-support systems address is proposing an *optimal* solution: a solution most likely to be accepted by all those involved in the multiuser scenario. Although an optimal solution may not exist for each multiuser scenario, identifying one, when it exists, can minimize the burden on the users to resolve the conflict manually.

Other researchers, e.g., [15, 51, 112, 115, 119], have proposed methods to automatically determine solutions to conflicts based on users' privacy preferences. These methods suffer from two main limitations. One, they always aggregate preferences in the same way regardless of the context or they only consider a very limited number of potential situations. Two, they do not consider the reasons behind users' preferences. However, evidence based on self-reported data [65, 131] suggests that users do listen to the explanations of others and that the optimal solution may depend on the particular context and reasons behind users' preferences. Following this idea, we empirically study three types of factors that potentially influence a privacy decision: the scenario's *context*, users' *preferences*, and their *arguments* about those preferences. An argument is a justification a user employs to convince the others involved that the user's expectations are reasonable and should be taken into account for making a decision.

Our key objective in this paper is to build an argumentation-based model that accurately

represents a multiuser scenario. To this end, we (1) identify important factors that potentially influence the inference; (2) evaluate the relative importance of these factors in inferring an optimal solution; and (3) develop a computational model that predicts an optimal solution for a given multiuser scenario.

We design a study where human participants are asked to choose what they think is the most appropriate sharing policy in a multiuser scenario we specified. Combining different values of the three types of factors we identified (context, preferences, and arguments), we generate 2,160 scenarios. Considering the sheer number of participants required, we conducted our study on Amazon Mechanical Turk (MTurk), collecting responses from 988 unique MTurk participants.

Contributions

(1) We propose a novel model for representing and reasoning about multiuser scenarios, employing three types of features: contextual factors, user preferences, and user arguments.

(2) We describe a crowdsourcing approach to build a training set for machine learning our model.

(3) We compare different classification techniques, within our model, finding that Random Forests yield the best accuracy (86%) in predicting the optimal sharing policy. We also evaluate the influence of different features on the optimal policy via a series of regression models.

6.1 Factors Influencing a Multiuser Sharing Decision

We identify three important classes of factors that potentially influence the privacy decision in a multiuser scenario: context, user preferences, and arguments by users.

6.1.1 Context

Following Dey et al. [20], we define the *context* of a multiuser scenario as any information—the information being shared or the people involved—that can influence the sharing decision. We identify four elements of the context.

Relationships among the individuals The type of relationship is known to play a crucial role when making *individual* decisions about privacy in social media [56, 74], specifically, people share information differently with friends, family, and colleagues. We hypothesize that the types of relationships influence how a person negotiates in a multiuser scenario, as attributes of a relationship such as intimacy may influence how much an opinion is taken into account. For example, following Wisniewski et al. [131], we imagine that a user's friend would respect the user's preferences.

Sensitivity of the information The sensitivity of the information to be shared is known to make an important contribution when making *individual* sharing decisions in social media [109, 125]. Besides, judgments of appropriateness of information to be shared on an SNS are subjective. For example, in some cultures, drinking alcohol is taboo. Thus, a photo showing a person drinking can be inappropriate in such a culture, whereas it can be normal in other cultures. Each individual involved in a scenario has a perception of the sensitivity of the information. This perception may affect how important that person thinks his view is on the appropriate sharing decision.

Sentiment of the information Personal information may evoke certain sentiments. For example, a photo from a birthday party where people are having fun evokes a positive feeling. Conversely, a photo from a funeral would typically evoke a negative feeling. As Kairam et al. [56] describe, the most common motivation to share information on an SNS is self-presentation: users care about what image they project. Therefore, the sentiment conveyed by the information they share can be important when deciding what to share.

6.1.2 Preferences

The goal of each individual in a multiuser scenario is to persuade the sharer to apply the individual's preferred sharing policy to the information. The individuals' (including sharer's) preferences can be compatible or conflicting. For example, Bob's preference of "I do not want my parents to see this photo" is compatible with Alice's preference of "I want only my friends to see it," as long as Bob's parents are not Alice's friends. Optimally, the final decision to resolve a conflict should respect the preferences of every individual involved to the best possible extent. If all preferences are compatible with each other, the solution is trivial. However, in case of conflict, an acceptable solution may not be evident.

A sharing policy can imply no sharing, sharing publicly, or anything in between. Further, depending on the number of contacts and their type, sharing policies change from one SNS user to another. Given the large space of possibilities, for the sake of simplicity, we consider the possible sharing policies to three levels of disclosure:

(1) *Share with all*: Anyone on the SNS can access the information.

(2) *Share with common friends*: Only common friends of the individuals involved in the scenario can access the information.

(3) *Share among themselves*: Only the individuals directly connected with the information can access the information.

6.1.3 Arguments

An individual involved in a multiuser scenario may employ arguments to convince the others that his preferred sharing policy should be used, or at least considered for making the final decision. There are potentially many arguments one can employ to negotiate with or persuade another. Arguments can be thought of as instances of argumentation schemes [123], representing forms of inference from premises to a conclusion. Walton et al. show that arguments used in everyday conversation fall into a small number of schemes.

We identify four argument schemes that can be effective in deciding an optimal solution for a multiuser scenario: *argument from* (i) *good consequences*, (ii) *bad consequences*, (iii) *an exceptional case*, and (iv) *popular opinion*. It is important to note that we neither claim these as the only possible schemes applicable in resolving a multiuser privacy conflict nor do we seek to evaluate if these are the best possible schemes. Our objective is to evaluate if arguments, as instances of schemes, help in deciding the final policy in multiuser scenarios. Our purpose of using argument schemes is to restrict the arguments to be of a few well-defined types, instead of choosing the arguments arbitrarily.

The general structure and examples for the argument schemes we use is as follows.

Argument from good or bad consequences *If A is brought about, then good (bad) consequences will occur. Therefore, A should (not) be brought about.*

An example of an argument from good consequences is: *We had a lot of fun during the party. Everybody's talking about how funny you were and they want to see your photos. Let's share it with everybody.* An example of an argument from bad consequences is: *It's a funny photo, but embarrassing since I appear drunk. I don't want strangers seeing it.*

An SNS user, who shares something, expects some benefit, e.g., friendship, jobs, or other social opportunities [25]. Thus, it is reasonable to argue that sharing certain information implies a good consequence. But, sharing inappropriate information can harm people's feelings and cause social tension. Thus, negative consequences can be valid arguments for not sharing.

Argument from an exceptional case *If the case of x is an exception, then the established rule can be waived in case of x.*

An example exceptional case arguments is: *C'mon! it was our graduation party! something that we do only once in our lifetimes. We should show it to the world.*

Although prior experience can guide future decisions, handling exceptions requires a different approach. Instances of this scheme cover cases where an unusual privacy configuration

is required: potentially, the opposite of the policy that might have been adopted if the arguments were not provided. Obviously, an individual must make a strong case to justify that the information is exceptional.

Argument from popular opinion *If a large majority in a particular group G accepts A as true (false), then there is a presumption in favor of (against) A.*

An example for a popular argument exception is: *The majority of the people that appear in the photo think that it should be kept private. Therefore, we should not share it with anyone.*

We consider that users never argue that the correct sharing policy is the one supported by the majority. For example, we do not consider arguments like "*The majority of us wants to share this photo only with common friends, hence, we should share it only with common friends.*" Instead, this argument *emerges* when two or more users suggest the same sharing policy. Thus, although no individual explicitly employed an argument from popular opinion, it is implicitly considered for the final outcome.

6.2 Inference Model

We describe a computational model that incorporates the factors identified in Section 6.1 to recommend an optimal sharing policy for a multiuser scenario. Figure 6.1 shows an overview of the model.

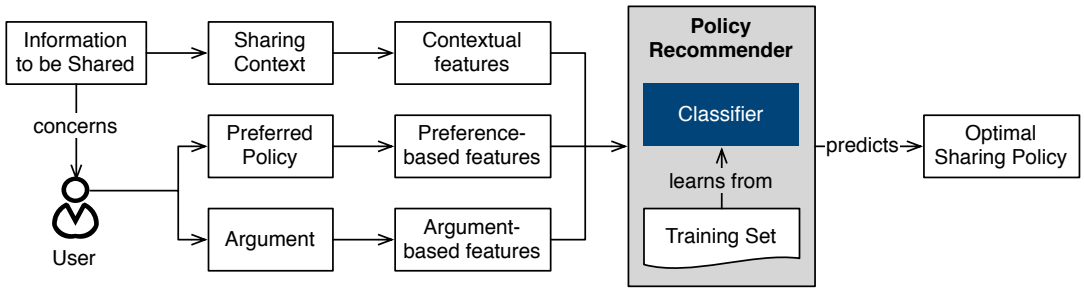


Figure 6.1: An overview of the inference model that predicts optimal sharing policy in a multiuser scenario

Given a piece of information to be shared in a multiuser scenario, we first identify all users involved in the scenario. This can be done, for example, by enabling a tagging mechanism. We then ask each user in the scenario to provide their preferred sharing policy for the information and an argument for why they want that preference. As for the sharing context: (1) relationships can be obtained directly from the SNS; (2) users can be asked to rate the sensitivity of the information; and (3) several automated approaches exist for computing the sentiment of the information.

The crux of the model is a *policy recommender*, which employs a machine learned *classifier* to recommend an optimal sharing policy for a given scenario. The recommender can employ a traditional classifier (we compare different techniques in Section 6.4.4). However, two important challenges are (1) to engineer the features of the classifier based on the sharing context, and the preferences and arguments of the users involved, and (2) to build a dataset to train and test the classifier.

Next, we describe a crowdsourcing approach to build a dataset for training the policy recommender. This approach is useful not only for testing the recommender, but also for building a seed training set required for practical deployment of the recommender. Once the recommender is in use, additional instances can be added to the training set as users share information via the model by adopting (or discarding) the recommender's suggestions.

6.2.1 Data Collection via Crowdsourcing

The dataset we seek must consist of a variety of multiuser scenarios and the optimal sharing policy in each of those scenarios. However collecting such data from users is challenging. For example, users are often reluctant to share sensitive information (one of the contextual factors in our model), biasing the study toward nonsensitive issues [125]. An alternative is asking users to self-report how they behave when they experience a multiuser scenario, but the results may not match participants' actual behavior because of the well-known dichotomy between users' stated privacy attitudes and their actual behavior [1]. Therefore, we chose to create *situations* in which

participants are *immersed* [73] to improve actual behavior elicitation while avoiding biasing the study to nonsensitive situations. We present information about two or more individuals in a specific circumstance: a combination of context, preferences, and arguments. We ask subjects to choose an optimal sharing policy for that circumstance.

We recruited participants for our study from Amazon MTurk [90]. We directed each participant to an external website that asked the participant to complete seven survey instruments: a presurvey questionnaire about demographics, five picture surveys (each involving a privacy conflict scenario and three sets of questionnaires), and a post-survey questionnaire about the participant's general opinions about resolving multiuser privacy conflicts. The study was approved by the IRB at North Carolina State University (details about participants and rewards in Section 6.2.1).

Presurvey Questionnaire

We asked participants to report their age, gender, level of education, how frequently they use social media, and how often they share (multiuser) pictures online. Since some of the situations we presented to the participants could be inappropriate for young readers, we required participants to be older than 18 years of age and showed a disclaimer at the beginning that the survey may be inappropriate for some users.

Picture Survey

The picture survey is the core of our study. We first show a picture and describe a hypothetical scenario in which the picture was taken and next ask a series of questions. Table 6.1 shows two examples of picture survey. We generated these and several similar picture surveys by combining factors identified in Section 6.1, as described below.

(1) Regarding context variables, we consider a predefined set of relation types, namely, *friends*, *family*, and *colleagues*. Further, we assume that all individuals involved in a scenario have the same type of relationship with each other (i.e., all of them are either *friends*, *family*, or

Table 6.1: Two example picture surveys (shortened version of those we used)

Picture		
Description	Aiko (C) took the picture above with her colleagues Ichiro and Katsu and, a French volunteer at the tsunami relief center	Three friends, Mark, Alex, and John, took the picture above during Mark's bachelor party on a boat in Ibiza
Rating	Identify the relationship between Aiko, Ichiro, and Katsu and rate the sensitivity and sentiment of the picture	Identify the relationship between Mark, Alex, and John and rate the sensitivity and sentiment of the picture
Context	Consider that Aiko wants to upload this picture to his social media account. What sharing policy should she apply for the picture?	Consider that Alex wants to upload this picture to his social media account. What sharing policy should he apply for the picture?
Preferences	Next, consider users' preferences as follows **Aiko** Share among ourselves **Ichiro** Share among ourselves **Katsu** Share with all Considering the context and users' preferences, what sharing policy should Aiko apply for the picture?	Next, consider users' preferences as follows **Mark** Share among ourselves **Alex** Share with common friends **John** Share with all Considering the context and users' preferences, what sharing policy should Alex apply for the picture?
Arguments	Finally, consider the users' preferences and arguments as follows **Aiko** This was one of the worst natural disasters ever. Share among ourselves **Ichiro** Tsunami was a disaster and our gestures are not appropriate; people may get the wrong idea. Share among ourselves **Katsu** The picture shows the difficult situation of survivors; sharing this can encourage people to help. Share with all Considering the context, and users' preferences and arguments, what sharing policy should Aiko apply for the picture?	Finally, consider the users' preferences and arguments as follows **Mark** There were some girls at the party; people might understand things the wrong way. Share among ourselves **Alex** This was one of the best day of our lives. Share with common friends **John** The is not like any other picture; it was from Mark's bachelor party! Share with all Considering the context, and users' preferences and arguments, what sharing policy should Alex apply for the picture?

colleagues). Also, the photos shown in the situations could be sensitive or nonsensitive and convey either a positive or a negative sentiment. This leads to 12 possible contexts. We

found a representative picture for each of those combinations. For example, in Table 6.1, for the picture on the left the relationship is *colleagues*, sensitivity is *low*, and sentiment is negative; in contrast, for the picture on the right the relationship is *friends*, sensitivity is *high*, and sentiment is positive.

(2) Some combinations of argument and privacy policy would not make sense in a real situation. Specifically, an *argument from bad consequence* does not often support a *share with all* policy. Similarly, an *argument from good consequence* does not often support a *share among themselves* policy. Further, we restrict ourselves to scenarios where *arguments from exceptional case* only support policies at either extreme: *share with all* or *share among themselves*. This yields six policy-argument combinations.

(3) We limit the number of individuals involved in each scenario to three. This way, the implicit *argument from popular opinion* could work (when applied) without ties. Although some scenarios we employed showed pictures with more than three individuals, our scenarios discussed the preferences and arguments of only three of the individuals among the people involved with the picture.

(4) We make sure that not all three individuals in a scenario use the same policy-argument combination. Our objective is to understand how a user decides a final policy given the scenario, and the preferences and arguments of others in the scenario. If all users thought the same way and wanted the same result, the solution would be trivial.

Putting the above together, we have: 12 pictures based on context, six policy-argument combinations each of first two individuals can employ, and five policy-argument combinations the last individual can employ (last restriction above). That is, we generated 2,160 scenarios. Each MTurk participant was shown five unique scenarios, making sure that no participant is shown the same picture twice. Further, we asked participants to immerse themselves in the particular scenario and ignore the resemblance or lack of resemblance of each scenario to other scenarios in which they might have seen this picture.

Following the picture and its description were four sets of questionnaires. We asked participants to answer these questionnaires sequentially and when answering a questionnaire, to consider consider information provided to them up to that point only.

The first questionnaire asks participants to assign values to three contextual variables: sensitivity (Likert scale 1 = not sensitive at all, 5 = very sensitive), sentiment (1 = extremely positive, 5 = extremely negative), and type of relationship among the people involved in the conflict (family, colleagues, or friends).

The next three questionnaires tell participants that one of the individuals in the scenario wants to upload the picture to a social media account. We ask participants what sharing policy should be applied. The participants choose one of the policies from *share with all*, *share with common friends*, and *share among themselves*. In the first case, participants know only the contextual attributes, but not the preferences or arguments of the individuals in the scenario. This case is similar to a real scenario where a user wants to upload and share information without asking others potentially concerned with the information. The second case introduces the preferences of all the users, but without their arguments supporting preferences. The third case employs all of the elements: the individuals in the scenario expose their preferences and support them with arguments. We keep the preferences fixed from the second case to the third, so we can observe the effect arguments have on the final decision.

Post-Survey Questionnaire

The post-survey questionnaire asks the following questions.

(1) How important do you think the following factors are in choosing an appropriate policy when sharing information concerning multiple users on social media? (a) Relationship between stakeholders; (b) Sensitivity of the information shared; and (c) Sentiment of the information shared. The response to each factor was on a Likert scale (1 = not important at all, 5 = extremely important).

(2) How confident will you be in choosing an appropriate policy for sharing information

concerning multiple users on social media in the following cases? (a) You do not know stakeholders preferences or arguments; (b) You know users preferences, but not their arguments; and (c) You know users preferences and arguments. The response to each case was on a Likert scale (1 = not confident at all, 5 = extremely confident).

Responses to the above questions allow us to find correlations (or lack thereof) between participants' self-reported behavior and what they actually answered during the study.

Participants and Quality Control

We needed 432 participants to receive one response per scenario (each participant responds to five of 2,160 scenarios). We intended to get two responses per scenario for completeness. However, we anticipated that some participants would begin a survey but not finish it, leaving gaps in the completed responses. To address this challenge, we launched the study on MTurk in multiple batches. For each batch, we checked if a particular survey needed additional responses and restricted the posted tasks accordingly. The final number of unique participants that completed the study was 988. At the end, each scenario had received at least two responses and some three responses. Compensation was provided for only those who completed all seven steps in the survey.

For quality control, we required participants to have completed at least 50 tasks on MTurk and to have had a success rate of at least 90% [92]. We also included an attention check question in the ratings section of each picture survey, asking how many people (faces) were present in the picture, answering which requires counting from the picture.

Table 6.2 summarizes our participants' responses to the presurvey questionnaire. The question corresponding to the last row in the table was in the post-survey questionnaire, so that participants understand what multiuser conflicts look like before answering that question. As shown, the majority of our participants used social media on a daily basis. Over 80% of our participants had shared a picture showing multiple users and about one-third of them had experienced privacy conflicts.

Table 6.2: Demographics of MTurk participants of our study

Gender	Male: 46.3%, Female: 53.4%, Other: 0.3%
Age	18–20: 2%, 21–29: 36.6%, 30–39: 36%, 40–49: 13.7%, 50–59: 7.5%, 60 or more: 4.1%
Education	Graduate degree: 11.2%, Bachelor degree: 44.4%, College no degree: 30.9%, High school: 12.4%, Less than high school: 1%
Social media usage	Daily: 83.9%, Weekly: 12%, Monthly: 3.7%, Never: 0.4%
Pictures shared	Many (>5): 35.1%, Few (1–5): 45%, None: 18.1%, Not sure: 1.7%
Conflicts experienced	Many (>5): 2.8%, Few (1–5): 30.1%, None: 66%, Not sure: 1.1%

6.2.2 Building a Training Set

We map the data collected in the MTurk study to a training set the policy recommender learns from. A training set consists of a set of data instances, where each data instance consists of a response variable (class) and a set of predictors (features). We set these as follows.

- A data instance corresponds to a participant's response to a picture survey.
- The response variable is the final policy chosen by the participant.
- The predictors are divided into three cases based on the picture survey. The first case consists of contextual features; the second case consists of contextual and preference-based features; and the third case consists of contextual, preference-based, and argument-based features. For brevity, we refer to these cases as Context, Preferences, and Arguments, respectively.

Contextual Features

We compute the contextual features based on participants' responses in the ratings section of the picture survey.

(1) *Sensitivity* and *sentiment* each yield a feature with five levels corresponding to ratings 1–5.

(2) *Relationship* yields a feature with three levels: *family*, *friendship*, and *colleagues*.

Preference-Based Features

We compute the following features based on the preferences portrayed in the corresponding scenario (picture survey). For concreteness, we base our examples on Table 6.1 (left).

(1) *Preference counts* represents three features corresponding to the number of participants preferring each of the three policies. In Table 6.1 (left), the preference counts are: *share with all* is 1, *share with common friends* is 0, and *share among themselves* is 2.

(2) *Most and least restrictive policies* represent, among the preferred policies of the users in a scenario, the policy restricting sharing of information the most and the least, respectively. The order of policies from least to most restrictive is: *share with all*, *share with common friends*, and *share among themselves*. In Table 6.1 (left), the most restrictive policy is *share among themselves*, and least restrictive policy is *share with all*.

(3) *Majority policy* represents the policy preferred by the majority of the users involved in the scenario. This feature can be *null* if there is no majority. The majority policy in Table 6.1 (left) is *share among themselves*.

Argument-Based Features

Unlike preference-based features, argument-based features incorporate preferences within an argument, each of which is an instance of one of the argument schemes we consider.

(1) *Argument counts* for Table 6.1 (left) are as follows. Each of these arguments has a count of 1: Aiko's *argument from an exceptional case* supporting *share among themselves*; Ichiro's *argument from negative consequence* supporting *share among themselves*; and Katsu's

argument from positive consequence supporting *share with all*. Each of the remaining three arguments (recall that there are six argument-policy combinations) has a count of 0.

(2) *Argument supporting least restrictive policy* is *argument from positive consequence* supporting *share with all*.

(3) *Arguments supporting least restrictive policy* are *argument from positive consequence* and *argument from an exceptional case* both supporting *share among themselves*.

(4) *Arguments supporting majority policy* are *argument from positive consequence* and *argument from an exceptional case* both supporting *share among themselves*.

When arguments from distinct schemes support a policy, as in the arguments supporting *least restrictive policy* and *majority policy* cases above, we employ the combination of arguments as a distinct feature value. Also, if a majority policy does not exist, we set the corresponding argument-based feature to *null*.

6.3 Evaluation

In this section, we describe our hypotheses and the techniques we use to evaluate those hypotheses.

6.3.1 Hypotheses

(1) *H-Influence-Context*: Contextual factors, specifically, sensitivity and sentiment of the information shared, and the relationships among individuals involved influence the optimal sharing policy in a multiuser scenario.

(2) *H-Influence-Preferences*: Preferences of users involved in a multiuser scenario influence the choice of optimal policy for the scenario.

(3) *H-Influence-Arguments*: Users' arguments for their preferred sharing policies influence the choice of optimal policy in a multiuser scenario.

(4) *H-Prediction-Preferences*: Adding preferences to the contextual factors enhances the accuracy of an inference model predicting the optimal policy for a given multiuser scenario.

(5) *H-Prediction-Arguments*: Adding arguments to preferences and contextual factors enhances the accuracy of an inference model predicting the optimal policy for a given multiuser scenario.

(6) *H-Confidence-Preferences*: Adding preferences to the contextual factors enhances a user's confidence in choosing the optimal policy for a given multiuser scenario.

(7) *H-Confidence-Arguments*: Adding arguments to preferences and contextual factors enhances a user's confidence in choosing the optimal policy for a given multiuser scenario.

6.3.2 Evaluation Strategy

To evaluate our hypotheses about influences of different factors (*H-Influence-*$*$), we build *multinomial logistic regression* models (multiple predictors and one response variable). We adopt the deviance of the fit as a measure of goodness of fit of these models. The deviance of fit is twice the difference between the maximum achievable log likelihood and that attained under the fitted model:

$$D = 2\sum_{i}^{n}\sum_{j}^{k} y_{ij} \log\left(\frac{y_{ij}}{\hat{\pi}_{ij} m_i}\right), \tag{6.1}$$

where π_{ij} is the categorical, cumulative, or conditional probability, and m_i is the corresponding sample size. Lower values of D indicate better fit.

Further, we focus on *coefficients* and their statistical significance. These coefficients help us understand each feature's relative influence on the optimal policy. For all models we employ *share among themselves* as the reference category. Thus, for a given category, positive coefficients

show how many times a variance in the level of a predictor increases the probability that the optimal policy is the given category. In the same fashion, negative coefficients indicate that increases in the level of the predictor increase the possibility that the optimal policy is the reference category.

We include all the features of a type (contextual factors, preferences, and arguments), when possible. However, some features of the same type can be highly correlated, causing *multi-collinearity* [36]. In that case, the coefficients may change erratically in response to small changes in the data. To counter this effect, we create independent models for highly correlated predictors. It is worth noting that this correction usually leads to higher coefficients.

To evaluate our hypotheses about prediction (*H-Prediction-*$*$), we build multiple *classification* models. We evaluate the prediction accuracy of these models via:

$$\begin{aligned} \text{precision} &= \frac{\text{TP}}{\text{TP} + \text{FP}}, \\ \text{recall} &= \frac{\text{TP}}{\text{TP} + \text{FN}}, \\ F_1\text{-measure} &= 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}, \end{aligned} \tag{6.2}$$

where TP, TN, FP, and FN refer to true and false positives and negatives. We implemented these models using Weka [38]. We perform 10-fold cross-validation and cross-validated paired t-test [21] to test significance in differences, when required.

To evaluate our hypotheses about users' confidence (*H-Confidence-*$*$), we perform the Kruskal-Wallis test [48] on self-reported data from the post-survey questionnaire. The Kruskal-Wallis test is a nonparametric extension (thus, does not require the assumptions that populations have normal distributions) of one-way ANOVA. This test compares medians to determine if all ratings come from the same distribution. If Kruskal-Wallis test determines that all the ratings do not come from the same distribution, we perform the multiple comparison test in Matlab [75] (with default settings) to determine which variables are significantly different.

6.4 Results

In this section, we evaluate each of our hypotheses.

6.4.1 Context (*H-Influence-Context*)

Table 6.3 shows the coefficients of the multinomial logistic regression model, employing contextual factors as predictors. We highlight statistically significant differences at significance levels (α) of 5% and 1% with * and **, respectively.

In the models for all three cases, we find that sensitivity is a significant influential factor. For example, in the first case, where only contextual factors are considered, the estimated coefficient -4.358 indicates that the probability of the optimal sharing policy being *share with all* compared to the probability of being *share among themselves* decreases exp(4.358) times for each increase in the sensitivity level of the photo, given all else equal. The model also points out that the significance of the type of the relationship decreases noticeably when preferences and/or arguments are considered. Actually, in Preferences and Arguments cases, relationship is not statistically significant on the relative probability of the optimal policy being *share with all* versus *share among themselves*. Finally, sentiment is not significant in any case. Our intuition, based on these observations, is that a highly sensitive picture is usually shared only among the individuals involved.

Table 6.3: Regression coefficients for contextual variables

Variable	Coefficients					
	Context		Preferences		Arguments	
	All	Common	All	Common	All	Common
Sensitivity	-4.358**	-3.154**	-1.479**	-1.18**	-1.274**	-1.104**
Sentiment	-0.194	-0.103	-0.179	0.099	-0.11	0.081
Relationship	0.441**	1.089**	0.16	0.412**	0.096	0.469**
D	.00765		.00793		.00799	

Further, we observe that as additional factors are considered, the coefficients of the contextual factors decrease and D increases. This indicates that, when preferences and arguments are taken into account by users, the contextual factors' influence on the optimal sharing policy diminishes.

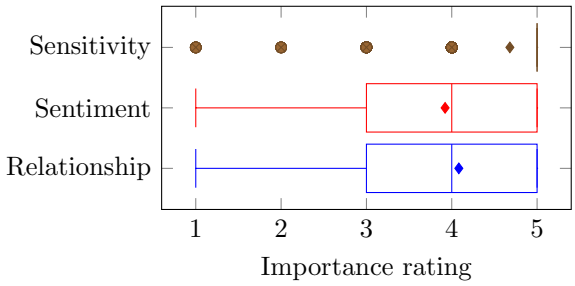


Figure 6.2: Importance (from the post-survey questionnaire) of context variables

To further analyze the contextual factors, we compare these results with self-reported data collected in the post-survey questionnaire. In this way, we can observe if participants' opinions are consistent with the decisions they made in the picture survey. Figure 6.2 shows the boxplots for participants' importance ratings to each contextual factor. A diamond dot in the boxplot indicates the mean. Each dot outside a box indicates an outlier.

From Table 6.3 and Figure 6.2, we observe that the regression model and self-reported values follow a similar pattern: sensitivity is the most influential factor. However, an important distinction between the two is that although relationship and sentiment have similar ratings in the post-survey questionnaire, sentiment is not significantly influential in the regression model. Also, in the post-survey questionnaire relationship and sentiment have high (median of 4) importance ratings. However, their low regression coefficients suggest that participants did not consider them as important as they said in the post-survey questionnaire.

6.4.2 Preferences (*H-Influence-Preferences*)

The logistic models for contextual factors suggest that employing preferences can influence the final sharing decision. Table 6.4 shows the coefficients for models employing preference-based features. Since Context does not include preferences, only Preferences and Arguments are shown. We note that preference counts are highly correlated: when one count increases, the other two (naturally) decrease, causing multicollinearity. To counter this, we create independent models for highly correlated predictors. Thus, the coefficient values for the preference counts features are obtained from different regression models: each of these models only employs one preference count as predictor.

Table 6.4: Regression coefficients for preference features

Variable	Coefficients			
	Preferences		Arguments	
	All	Common	All	Common
# Share with all	1.13**	0.595**	1.18**	0.65**
# Share with common friends	0.163*	0.925**	0.13	0.874**
# Share among themselves	−0.991**	−1.266**	−1**	−1.273**
Least restrictive policy	−1.524**	−0.043	−1.822**	−0.308
Most restrictive policy	−3.237**	−4.499**	−2.929**	−4.16**
Majority policy	−0.694**	−0.739**	−0.625**	−0.816**
D	.00728		.00734	

First, we observe that the most restrictive policy has the most influence on the final policy, more so than even the majority policy. Second, we observe that preference counts have, in almost every case, a statistically significant influence on the final sharing policy. Finally, the coefficients of the preference-based features do not change much from Preferences to Arguments. This indicates that preferences remain important even when both preferences and arguments are considered.

6.4.3 Arguments (*H-Influence-Arguments*)

Table 6.5 shows the regression coefficients for the argument-based features. As for the policy counts, the argument counts are also highly correlated. Thus, we use different models for those features.

Considering the absolute values of the coefficients (ignoring the sign), preferences supported by exceptional and positive arguments have the highest influence in the optimal sharing policy. On the other hand, the arguments for negative consequences yield the lowest coefficients. This suggests that users tend to value the benefits more than the risks of sharing a picture. However, it is worth noting that *argument from bad consequences* supporting *self* (share among themselves) has the highest coefficient of all argument counts. This indicates that when a user makes a strong case for not sharing a picture, the other users respect that preference. This finding is consistent with those of Besmer and Lipford [8], and Wisniewski et al. [131].

Table 6.5: Regression coefficients for argument features

Variable	Coefficients	
	All	Common
# Positive supporting all	1.528**	0.927**
# Positive supporting common	0.536**	1.282**
# Negative supporting common	−0.169	1.05**
# Negative supporting self	−2.139**	−2.675**
# Exceptional supporting all	1.061**	0.627**
# Exceptional supporting self	−0.833**	−1.18**
Positive supporting least restrictive policy	−0.89**	−1.049**
Negative supporting least restrictive policy	−1.4**	−0.99**
Exceptional supporting least restrictive policy	−1.144**	−0.928**
Positive supporting most restrictive policy	1.554**	1.16**
Negative supporting most restrictive policy	0.948**	1.085**
Exceptional supporting most restrictive policy	1.328**	1.207**
Positive supporting majority policy	2.19**	1.475**
Negative supporting majority policy	−0.094	0.635**
Exceptional supporting majority policy	−2.001**	−2.143**
D	.00739	

6.4.4 Prediction (*H-Prediction-Preferences* and *H-Prediction-Arguments*)

The foregoing hypotheses concerned how various features influence a sharing policy. Now, we evaluate a predictive model that puts these features to use. Since our objective is to predict the actual policy (*all*, *common*, or *self*), we build three-class classifiers, which make discrete predictions (in contrast to regression models, which make continuous predictions).

Table 6.6: Accuracies of decision tree classifiers for the three cases, considering all data instances

	Precision	Recall	F_1	Class	Count
Context	0.521	0.463	0.490	*all*	1032
	0.493	0.420	0.454	*common*	1405
	0.590	0.720	0.649	*self*	1463
	0.537	**0.544**	**0.537**	weighted mean	–
Preferences	0.318	0.047	0.081	*all*	601
	0.591	0.609	0.600	*common*	1590
	0.606	0.772	0.679	*self*	1709
	0.556	**0.594**	**0.555**	weighted mean	–
Arguments	0.326	0.100	0.153	*all*	628
	0.614	0.641	0.627	*common*	1528
	0.645	0.780	0.706	*self*	1744
	0.581	**0.616**	**0.586**	weighted mean	–

To start with, we build *decision tree* classifiers, using the J48 implementation in Weka [38], for different feature sets. Table 6.6 compares the prediction accuracies of decision trees for the three cases. The count column in the table shows, for each class, the count of the actual (ground truth) instances belonging to that class.

First, we observe that the F_1 measure increases (1) from Context to Preferences: contextual features to contextual and preference-based feature (*H-Prediction-Preferences*), and (2) from Preferences to Arguments: contextual and preference-based features to contextual, preference- and argument-based features (*H-Prediction-Arguments*). Further, from a 10-fold cross-validated

paired t-test [21], we find that the differences in the F_1 measures are also significant ($p < 0.01$).

We observe that the F_1 measures in Table 6.6, although consistent with our hypotheses, are not impressive. Investigating this further, we trace the major reason for a low F_1 measure to participants providing inconsistent responses. Recall that each of our scenarios received responses from two participants (and some from three). In many cases, the final policy chosen by different participants was not same. In such cases, irrespective of what we predict, the prediction cannot be accurate for all instances. To counter this, we next considered instances corresponding to only to consistent responses.

Table 6.7: Accuracies of decision tree classifiers for the three cases, considering consistent data instances only

	Precision	Recall	F_1	Class	Count
Context	0.577	0.507	0.540	*all*	361
	0.590	0.462	0.518	*common*	504
	0.693	0.854	0.765	*self*	657
	0.631	**0.642**	**0.630**	weighted mean	–
Preferences	0.000	0.000	0.000	*all*	122
	0.778	0.703	0.739	*common*	794
	0.724	0.885	0.797	*self*	881
	0.699	**0.745**	**0.717**	weighted mean	–
Arguments	0.508	0.246	0.332	*all*	134
	0.808	0.823	0.816	*common*	707
	0.843	0.894	0.868	*self*	916
	0.803	**0.816**	**0.806**	weighted mean	–

Table 6.7 compares the prediction accuracies for the three cases, considering only the consistent responses. Now, we observe that not only do the accuracies increase for each case, the differences in the accuracies of the cases increase as well.

Next, we make two interesting observations from counts in Tables 6.6 and 6.7. First, we observe that the count for *all* class reduces drastically from Context to Preferences or Arguments. Thus, given preferences or arguments, users tend to choose more restrictive policies. Second, the

count for *common* class reduces from Preferences to Arguments. This suggests that given only preferences, users tend to choose a safe option (*common*). However, supporting preferences by arguments can encourage users to choose extreme options in some scenarios (notice the increase in *self* and all classes for Arguments).

Table 6.8: Accuracies of three classifiers for Arguments, considering consistent data instances only

	Precision	Recall	F_1	Class
Naive Bayes	0.313	0.500	0.385	*all*
	0.838	0.666	0.742	*common*
	0.818	0.876	0.846	*self*
	0.787	**0.763**	**0.769**	weighted mean
Logistic Regression	0.456	0.351	0.397	*all*
	0.803	0.789	0.796	*common*
	0.836	0.876	0.855	*self*
	0.794	**0.801**	**0.797**	weighted mean
Random Forests	0.653	0.463	0.541	*all*
	0.875	0.849	0.861	*common*
	0.882	0.940	0.910	*self*
	0.862	**0.867**	**0.862**	weighted mean

Finally, to make sure that our conclusion are not specific to decision trees, we experiment with three more classifiers: Naive Bayes, Logistic Regression, and Random Forests. Table 6.8 compares the accuracies of these models for Arguments. We observe that Random Forests, an ensemble learning approach, yields the best accuracies among all the four classifiers (including decision trees).

6.4.5 Confidence (*H-Confidence-Preferences* and *H-Confidence-Arguments*)

We compare a user's confidence in choosing an optimal sharing policy given information corresponding to the three cases. Figure 6.3 shows the boxplots of the participants' confidence ratings collected from the post-survey questionnaire. Further, from Kruskal-Wallis and multi-

ple comparison tests, we find that the median ($\tilde{x}$) confidence rating for Preferences compared to Context, and Arguments compared to Preferences is significantly ($p < 0.01$) larger. This suggests that adding preferences and arguments to the contextual factors increases a user's confidence in choosing an optimal sharing policy for a scenario.

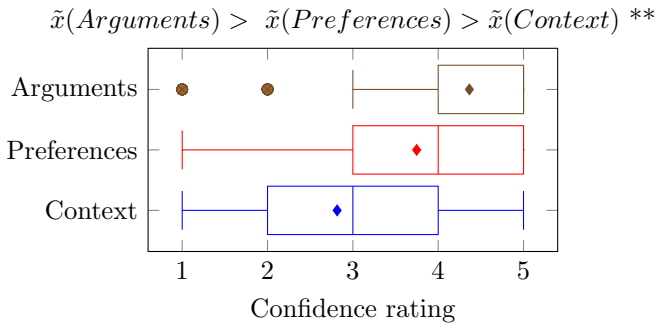


Figure 6.3: Users' confidence (from the post-survey questionnaire) in choosing the optimal policy for different cases

6.5 Discussion

Our results show that contextual factors, preferences, and arguments influence the optimal sharing policy. However, we find a disconnection between the importance rating assigned by participants to contextual factors and how these factors actually influence the optimal policy. Participants considered that the sentiment a picture conveys is an important element (rating 4 out of 5). However, according to our regression model, the influence of sentiment over the optimal policy is not statistically significant.

The inference models show that arguments help in predicting the optimal policy. Moreover, the majority of participants reported that they feel much more confident about what policy to apply to a shared item if they know others' arguments and preferences. These results indicate that approaches based only on aggregating user preferences, e.g., [15, 50, 119] (as discussed below), may not be effective. Our results show that users are willing to accommodate others'

preferences if they know their reasons. Therefore, knowing only the preferences is not enough to set a policy that is satisfactory for the majority of the users involved.

6.5.1 Threats to Validity

We identify two threats to validity of our findings. First, any study based on surveys is susceptible to participant confusion. We mitigated this threat via explicit quality control measures normally used in MTurk studies, such as number of previous tasks completed, task success rate and the use of attention check questions as detailed in Section 6.2.1. Second, to avoid the unreliability of self-reported attitudes, the well-known reluctance of participants to provide sensitive information, and the logistical challenges of finding participants in sets of three friends, we employ scenarios in which users immerse themselves and provide their assessment of the policies, preferences, and arguments under consideration. This is based on methodologies known to work well in other domains, such as [73], as well as methodologies used in other social media privacy studies, such as [43, 59, 127]. Generalisations, however, should be made with caution, as participant's decisions in immersive scenarios may not necessarily match those in real scenarios.

6.5.2 Limitations and Directions

We identify three directions for future work. First, for logistical reasons, we employ predefined types of arguments, policies, and contexts. Thus, our findings are specific to these values. Nonetheless, other schemes can fit in a multiuser scenario. A future study could attempt to understand what arguments users would employ to support their preferences during a multiuser scenario—benefit being to discover new argumentation schemes. Automated techniques for identifying arguments [70] can be valuable in this regard.

Second, our work contributes to inform the development of decision-support systems for multiuser scenarios by modelling context, preferences and arguments to find an optimal sharing policy. However, the are other challenges that would need to be addressed in order to develop a

usable user interface for any such decision-support system, such as finding an adequate trade-off between user intervention and automation to avoid becoming a burden on the users. To this aim, suitable defaults [126] or recommender systems based on machine-learning approaches [27] could be applied. As future work, we plan on building a recommender tool and test it with users, so we can collect information about its accuracy and usability.

Third, our study is cross-sectional (one time) and in the scenarios of the study, decisions are to be made in one round. Thus, our study does not provide data to understand how users deal with situations where their arguments are countered by others' arguments. A future study could require participants to engage in negotiation and persuasion to decide an optimal sharing policy for a scenario. Such a study requires coordinating multiple participants; thus, conducting such a study on MTurk is nontrivial. Employing automated agents (built based on the data we collected in our study) against one or a few real participants is a viable and interesting direction.

6.5.3 Related Work

Wisniewski et al. [131] showed a distinct lack of tools to manage multiuser scenarios in mainstream SNSs. They also showed this lack forces users to utilise a number of *coping strategies* with limited effectiveness in practice. These strategies include blocking other users, filtering friendship requests by the number of common friends, self censorship etc. In a similar previous study, Lampinen et al.[65] explore SNS-users' perceptions of control over online disclosure through a series of interviews. The qualitative data obtained in their study suggests that users manage interpersonal boundaries both individually and collaboratively. The authors further classify the strategies that SNS users employ to manage their privacy into two super classes: preventive and corrective. Strategies in both classes can be accomplished through individual or collaborative means. For example, a collaborative preventive strategy is asking for approval before disclosing content from those involved. These two studies show that SNS users need functionalities to manage multiuser privacy. Our work is a stepping stone towards offering such

functionality.

Besmer and Lipford [8] propose a method where the owner of a photograph in a multiuser scenario is in charge of deciding the sharing policy and the other involved users are able to suggest privacy preferences. In particular, the authors developed a Facebook application that allows a user to send privacy suggestions to the owner of a photo where that user appears. The authors show that the owners of the photos would most of the time listen to and consider the suggestions of the others to make a decision about sharing policies, which is consistent with our findings. However, the main issue with this approach is that it is all done in a *manual* way, i.e., it is the owner the one that needs to come up with the optimal solution after listening to others' petitions. Therefore, the owner of the photo may feel overloaded with the petitions of other users and the potential conflicts that could subsequently arise from attending to these petitions.

Thomas et al. [119] propose *veto voting* as a direct approach to manage multiuser sharing policies. That is, denying access takes precedence over granting access. Thus, if an individual wants to share the information with a given user, but another individual does not, the information is not shared. Whereas this approach does not allow privacy breaches, it may lead to utility loss. For example, suppose Alice and Bob appear together in a picture. Bob initially opposes sharing the picture with Charlie as he does not know him. However, if Alice tells him that Charlie is her friend and that everything is OK, then Bob may accept sharing with Charlie. Had veto voting been applied, the picture would have not been shared with Charlie, thereby missing, for example, a potential opportunity for Bob to be friends with Charlie.

Other approaches too tackle multiuser privacy conflicts [15, 51, 50, 112, 130]. However, some of these approaches require too much human intervention during conflict resolution: Wishart et al. [130] require users to solve the conflicts *manually* and Squicciarini et al. [112] do so nearly *manually*, e.g., by participating in difficult-to-comprehend auctions with fake money for every possible conflict. Approaches that provide automation [15, 50, 119] help resolve multiuser conflicts, but they consider only one fixed way of aggregating user preferences, without considering

how users would compromise and the concessions they might be willing to make in a specific situation. Hu and Ahn [51] consider more than one way of aggregating user preferences, but the user that uploads the item chooses the aggregation method to be applied, which becomes a unilateral decision without considering any input from others. Clearly, solutions that do not consider input from all users involved may lead to solutions that are far from what some users would be willing to accept. In essence, current automated automated mechanisms do not readily adapt to different situations that may motivate users' concessions, which has the potential to cause these mechanisms to suggest solutions that may not be acceptable to all concerned. This leads users to manually resolve conflicts most of the times. Such and Criado [114, 115] provide an improvement over the fixed ways of aggregating user preferences by suggesting three, but only three, methods to be selected depending on the situation.

Some recent works, e.g., [52, 117], propose game-theoretic mechanisms to tackle multiuser privacy conflicts. These proposals provide formal frameworks based on established concepts such as Nash equilibrium. However, Hu et al. [52] show, such proposals may not work well in practice since they may not capture the social idiosyncrasies that users consider in real life [65, 131].

Ilia et al. [53] present a mechanism to enforce fine-grained access control in photos by blurring the faces of the users depicted in the photo based on each users' access control list. This approach can limit the utility of sharing information. However, used in conjunction with a decision-support mechanism based our findings, Ilia et al.'s approach could enforce access control differently for different users in case they cannot agree on a sharing policy.

Murukannaiah et al. describe Danio [82], a methodology for engineering privacy in social applications. Their methodology relies on understanding privacy norms. Muppet provides a mechanism for eliciting privacy norms from the crowd.

Arguments are used to resolve conflicts is other domains. Murukannaiah et al. [84] employ arguments in requirements engineering to resolve conflicts in stakeholders' goals. Their findings that arguments lead to consistent responses aligns with our observation. Williams and

Williamson [128] incorporate arguments and Bayesian networks for breast cancer prognosis, where they exploit arguments to develop an explanation for the prognosis. A similar application in the privacy domain can be to explain to a user, via arguments, the consequences of a particular sharing decision.

6.6 Conclusions

Sharing all kinds of information on SNSs is routine for many people. Examples of such information are a photo from the Christmas party and a tweet about your imminent trip with friends. The information shared often involves multiple users. When the privacy preferences of two or more users do not align, they can negotiate to balance privacy and utility for each user. Related literature proposes methods based on aggregating privacy preferences. However, aggregation does not capture how people align with others' preferences. In contrast, we propose employing arguments to support preferences. We present an inference model that employs sharing context, and users' preferences and arguments to predict an optimal sharing policy in a multiuser scenario. We conduct a crowdsourcing study to train and test our model.

Via a series of multinomial logistic regression models, we show that all three feature types we consider influence the optimal sharing policy. We find that (1) sensitivity has the highest influence on the optimal policy, among contextual variables; (2) the most restrictive policy, not the majority policy, has the highest influence on the optimal policy, among preference-based features; and (3) users value arguments for sharing more than those for not sharing; however, if the argument for not sharing is an exceptional case argument users usually support not sharing.

By training an inference model for each feature type, we find that a model employing argument-based features predicts optimal policy with higher accuracy than those not employing arguments. Further, from self reported data, we find that adding arguments increases a user's confidence in choosing the final sharing policy. Finally, we compare different classification techniques, within our model, and find that Random Forests, when all features are considered, yields the highest (86%) accuracy in predicting the optimal sharing policy in a multiuser scenario.

REFERENCES

[1] Alessandro Acquisti and Ralph Gross. Imagined communities: Awareness, information sharing, and privacy on the facebook. In *Proceedings of the 6th International Conference on Privacy Enhancing Technologies (PET)*, pages 36–58, Cambridge, UK, 2006.

[2] Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini. Reasoning with contextual requirements: Detecting inconsistency and conflicts. *Information and Software Technology*, 55(1):35–57, January 2013.

[3] Android Open Source Project. Android Developers Guide: Obtaining user location. `http://developer.android.com/guide/topics/location/obtaining-user-location.html`, 2012.

[4] Daniel Ashbrook and Thad Starner. Learning significant locations and predicting user movement with GPS. In *Proceedings of the 6th International Symposium on Wearable Computers*, pages 101–108, Seattle, WA, 2002.

[5] Martin Azizyan, Ionut Constandache, and Romit Roy Choudhury. SurroundSense: Mobile phone localization via ambience fingerprinting. In *Proceedings of the 15th Annual International Conference on Mobile Computing and Networking (MobiCom)*, pages 261–272, Beijing, 2009.

[6] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, June 2007.

[7] Carole Bernon, Valérie Camps, Marie-Pierre Gleizes, and Gauthier Picard. Engineering adaptive multi-agent systems: The ADELFE methodology. In Brian Henderson-Sellers and Paolo Giorgini, editors, *Agent-Oriented Methodologies*, chapter 7, pages 107–135. Idea Group, Hershey, PA, 2005.

[8] Andrew Besmer and Heather Richter Lipford. Moving beyond untagging: Photo privacy in a tagged world. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, pages 1563–1572, Atlanta, 2010.

[9] Claudio Bettini, Oliver Brdiczka, Karen Henricksen, Jadwiga Indulska, Daniela Nicklas, Anand Ranganathan, and Daniele Riboni. A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, 6(2):161–180, April 2010.

[10] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, Secaucus, NJ, 2006.

[11] Cristiana Bolchini, Giorgio Orsi, Elisa Quintarelli, Fabio A. Schreiber, and Letizia Tanca. Context modeling and context awareness: Steps forward in the Context-ADDICT project. *IEEE Data Engineering Bulletin*, 34(2):47–54, 2011.

[12] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, May 2004.

[13] Senaka Buthpitiya, Faisal Luqman, Martin Griss, Bo Xing, and Anind K. Dey. Hermes – A context-aware application development framework and toolkit for the mobile environment. In *Proceedings of the 26th International Conference on Advanced Information Networking and Applications Workshops*, pages 663–670, Fukuoka, Japan, March 2012.

[14] Licia Capra, Wolfgang Emmerich, and Cecilia Mascolo. CARISMA: Context-aware reflective middleware system for mobile applications. *IEEE Transactions on Software Engineering*, 29(10):929–945, October 2003.

[15] Barbara Carminati and Elena Ferrari. Collaborative access control in on-line social networks. In *Proceedings of the 7th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, pages 231–240, October 2011.

[16] Stefano Ceri, Florian Daniel, Maristella Matera, and Federico M. Facca. Model-driven development of context-aware web applications. *ACM Transactions on Internet Technology*, 7(1):1–33, February 2007.

[17] Tanzeem Choudhury and Alex Pentland. Sensing and modeling human networks using the Sociometer. In *Proceedings of IEEE International Symposium on Wearable Computers*, pages 216–222, White Plains, NY, October 2003.

[18] William Deresiewicz. Faux friendship. *The Chronicle of Higher Education*, December 2009.

[19] Marie desJardins, Eric Eaton, and Kiri Wagstaff. A context-sensitive and user-centric approach to developing personal assistants. In *Working Notes of the AAAI Spring Symposium on Persistent Assistants*, pages 98–100, March 2005.

[20] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16(2):97–166, December 2001.

[21] Thomas G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923, October 1998.

[22] Matt Duckham and Lars Kulik. Location privacy and location-aware computing. In Jane Drummond, Roland Billen, Elsa João, and David Forrest, editors, *Dynamic and Mobile GIS: Investigating Changes in Space and Time*, chapter 3, pages 34–51. CRC Press, Boca Raton, FL, 2006.

[23] Henry Been-Lirn Duh, Gerald C. B. Tan, and Vivian Hsueh-hua Chen. Usability evaluation for mobile device: A comparison of laboratory and field tests. In *Proceedings of the 8th Conference on Human-Computer Interaction with Mobile Devices and Services*, pages 181–186, Helsinki, September 2006.

[24] Nathan Eagle and Alex (Sandy) Pentland. Reality mining: Sensing complex social systems. *Personal and Ubiquitous Computing*, 10(4):255–268, March 2006.

[25] Nicole B Ellison, Charles Steinfield, and Cliff Lampe. The benefits of Facebook "friends:" social capital and college students' use of online social network sites. *Journal of Computer-Mediated Communication*, 12(4):1143–1168, 2007.

[26] Wolfgang Emmerich. Software engineering and middleware: A roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 117–129, Limerick, Ireland, 2000.

[27] Lujun Fang and Kristen LeFevre. Privacy wizards for social networking sites. In *Proceedings of the 19th International Conference on World Wide Web (WWW)*, pages 351–360, Raleigh, North Carolina, 2010. ACM.

[28] Ricard L. Fogues, Pradeep K. Murukannaiah, Jose M. Such, Agustin Espinosa, Ana Garcia-Fornes, and Munindar P. Singh. Argumentation for multi-party privacy management. In *Second International Workshop on Agents and CyberSecurity (ACySe)*, pages 3–6. 2015.

[29] Ricard L. Fogués, Jose M. Such, Agustín Espinosa Minguet, and Ana García-Fornes. Open challenges in relationship-based privacy mechanisms for social network services. *International Journal of Human-Computer Interaction*, 31(5):350–370, 2015.

[30] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3–5):75–174, February 2010.

[31] John E. Freund and Benjamin M. Perles. *Statistics: A First Course*. Prentice Hall, Upper Saddle River, NJ, 2004.

[32] Thomas F. Gieryn. A space for place in sociology. *Annual Review of Sociology*, 26(1):463–496, August 2000.

[33] Lakshmi Goel, Norman A. Johnson, Iris Junglas, and Blake Ives. From space to place: Predicting users' intentions to return to virtual worlds. *Management Information Systems Quarterly*, 35(3):749–771, September 2011.

[34] Google. Google Places API. `https://developers.google.com/places/`, 2013.

[35] Mark S. Granovetter. The strength of weak ties. *American Journal of Sociology*, 78(6):1360–1380, May 1973.

[36] Damodar N. Gujarati and Dawn C. Porter. *Basic Econometrics*, chapter Multicollinearity: What happens if the regressors are correlated? McGraw-Hill, Irwin, 2009.

[37] Bin Guo, Daqing Zhang, and Michita Imai. Toward a cooperative programming framework for context-aware applications. *Personal and Ubiquitous Computing*, 15(3):221–233, March 2011.

[38] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations Newsletter*, 11(1):10–18, November 2009.

[39] Chung-Wei Hang, Pradeep K. Murukannaiah, and Munindar P. Singh. Platys: User-centric place recognition. In *AAAI Workshop on Activity Context-Aware Systems*, 2013.

[40] Sudheendra Hangal, Diana MacLean, Monica S. Lam, and Jeffrey Heer. All friends are not equal: Using weights in social graphs to improve search. In *Proceedings of the ACM Workshop on Social Network Mining and Analysis*, pages 1–7, July 2010.

[41] Ramaswamy Hariharan and Kentaro Toyama. Project Lachesis: Parsing and modeling location histories. In *Proceedings of the 3rd International Conference on Geographic Information Science*, pages 106–124, October 2004.

[42] Steve Harrison and Paul Dourish. Re-place-ing space: The roles of place and space in collaborative systems. In *Proceedings of the 10th ACM Conference on Computer Supported Cooperative Work*, pages 67–76, Boston, November 1996. ACM.

[43] Michael Hart, Claude Castille, Rob Johnson, and Amanda Stent. Usable privacy controls for blogs. In *Proceedings of the International Conference on Computational Science and Engineering*, volume 4, pages 401–408, Aug 2009.

[44] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, New York, 2001.

[45] Adel Hendaoui, Moez Limayem, and Craig W. Thompson. 3D social virtual worlds: Research issues and challenges. *IEEE Internet Computing*, 12(1):88–92, January 2008.

[46] Karen Henricksen and Jadwiga Indulska. Developing context-aware pervasive computing applications: Models and approach. *Pervasive and Mobile Computing*, 2(1):37–64, February 2006.

[47] Jeffrey Hightower, Sunny Consolvo, Anthony LaMarca, Ian E. Smith, and Jeff Hughes. Learning and recognizing the places we go. In *Proceedings of the 7th International Conference on Ubiquitous Computing*, pages 159–176. Springer, September 2005.

[48] Myles Hollander and Douglas A. Wolfe. *Nonparametric Statistical Methods*. Wiley, New York, 1999.

[49] Jongyi Hong, Eui-Ho Suh, Junyoung Kim, and SuYeon Kim. Context-aware system for proactive personalized service based on context history. *Expert Systems with Applications*, 36(4):7448–7457, May 2009.

[50] Hongxin Hu, Gail-Joon Ahn, and Jan Jorgensen. Detecting and resolving privacy conflicts for collaborative data sharing in online social networks. In *Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC)*, pages 103–112, Orlando, FL, 2011.

[51] Hongxin Hu, Gail-Joon Ahn, and Jan Jorgensen. Multiparty access control for online social networks: Model and mechanisms. *IEEE Transactions on Knowledge and Data Engineering*, 25(7):1614–1627, July 2013.

[52] Hongxin Hu, Gail-Joon Ahn, Ziming Zhao, and Dejun Yang. Game theoretic analysis of multiparty access control in online social networks. In *Proceedings of the 19th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 93–102, London, ON, 2014.

[53] Panagiotis Ilia, Iasonas Polakis, Elias Athanasopoulos, Federico Maggi, and Sotiris Ioannidis. Face/Off: Preventing privacy leakage from photos in social networks. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 781–792, Denver, Colorado, 2015. ACM.

[54] Valerie Issarny, Mauro Caporuscio, and Nikolaos Georgantas. A perspective on the future of middleware-based software engineering. In *Proceedings of the Conference on the Future of Software Engineering*, pages 244–258, Washington, DC, 2007. IEEE Computer Society.

[55] Natalia Juristo and Ana M. Moreno. *Basics of Software Engineering Experimentation.* Kluwer Academic Publishers, Boston/Dordrecht/London, 2001.

[56] Sanjay Kairam, Mike Brzozowski, David Huffaker, and Ed Chi. Talking in circles: Selective sharing in Google+. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, pages 1065–1074, Austin, 2012.

[57] Jong Hee Kang, William Welbourne, Benjamin Stewart, and Gaetano Borriello. Extracting places from traces of locations. *Mobile Computing Communications Review*, 9(3):58–68, July 2005.

[58] Donnie H. Kim, Younghun Kim, Deborah Estrin, and Mani B. Srivastava. SensLoc: sensing everyday places and paths using less energy. In *Proceedings of the 8th Conference on Embedded Networked Sensor Systems*, pages 43–56, Zurich, November 2010.

[59] Peter Klemperer, Yuan Liang, Michelle Mazurek, Manya Sleeper, Blase Ur, Lujo Bauer, Lorrie Faith Cranor, Nitin Gupta, and Michael Reiter. Tag, you can see it!: using tags for access control in photo sharing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, pages 377–386, 2012.

[60] Klaus Krippendorff. Reliability in content analysis. *Human Communication Research*, 30(3):411–433, 2004.

[61] Devdatta Kulkarni, Tanvir Ahmed, and Anand Tripathi. A generative programming framework for context-aware CSCW applications. *ACM Transactions on Software Engineering and Methodology*, 21(2):1–35, March 2012.

[62] Axel Küpper. *Location-Based Services: Fundamentals and Operation.* John Wiley and Sons, Chichester, UK, 2005.

[63] Jennifer R. Kwapisz, Gary M. Weiss, and Samuel A. Moore. Activity recognition using cell phone accelerometers. *SIGKDD Explorations*, 12(2):74–82, March 2011.

[64] Gerard Kyle and Garry Chick. The social construction of a sense of place. *Leisure Sciences*, 29(3):209–225, May 2007.

[65] Airi Lampinen, Vilma Lehtinen, Asko Lehmuskallio, and Sakari Tamminen. We're in it together: Interpersonal management of disclosure in social network services. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, pages 3217–3226, Vancouver, 2011.

[66] Maria Lewicka. Place attachment: How far have we come in the last 40 years? *Journal of Environmental Psychology*, 31(3):207–230, September 2011.

[67] Yang Li, Jason I. Hong, and James A. Landay. Topiary: A tool for prototyping location-enhanced applications. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*, pages 217–226, Santa Fe, NM, October 2004.

[68] Dekang Lin. An information-theoretic definition of similarity. In *Proceedings of the 15th International Conference on Machine Learning*, pages 296–304, July 1998.

[69] Jialiu Lin, Guang Xiang, Jason I. Hong, and Norman M. Sadeh. Modeling people's place naming preferences in location sharing. In *Proceedings of the 12th International Conference on Ubiquitous Computing (UbiComp)*, pages 75–84, Copenhagen, September 2010.

[70] Marco Lippi and Paolo Torroni. Context-independent claim detection for argument mining. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, pages 185–191, Buenos Aires, 2015.

[71] Carsten Magerkurth, Adrian David Cheok, Regan L. Mandryk, and Trond Nilsen. Pervasive games: Bringing computer entertainment back to the real world. *Computers in Entertainment*, 3(3):4–4, July 2005.

[72] Marco Mamei and Franco Zambonelli. Programming pervasive and mobile computing applications: The TOTA approach. *ACM Transactions on Software Engineering and Methodology*, 18(4):1–56, July 2009.

[73] Clara Mancini, Yvonne Rogers, Arosha K Bandara, Tony Coe, Lukasz Jedrzejczyk, Adam N. Joinson, Blaine A. Price, Keerthi Thomas, and Bashar Nuseibeh. ContraVision: Exploring users' reactions to futuristic technology. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, pages 153–162. ACM, 2010.

[74] Alice Marwick and Danah Boyd. I tweet honestly, I tweet passionately: Twitter users, context collapse, and the imagined audience. *New Media & Society*, 13(1):114–133, September 2011.

[75] Matlab. Multiple comparison test. `http://www.mathworks.com/help/stats/multcompare.html`, 2015.

[76] Thomas J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, July 1976.

[77] Melinda J. Milligan. Interactional past and potential: The social construction of place attachment. *Symbolic Interaction*, 21(1):1–33, February 1998.

[78] Raúl Montoliu and Daniel Gatica-Perez. Discovering human places of interest from multimodal mobile phone data. In *Proceedings of the 9th International Conference on Mobile and Ubiquitous Multimedia*, pages 12:1–12:10, December 2010.

[79] Amy L. Murphy, Gian Pietro Picco, and Gruia-Catalin Roman. LIME: A coordination model and middleware supporting mobility of hosts and agents. *ACM Transactions on Software Engineering and Methodology*, 15(3):279–328, July 2006.

[80] Pradeep K. Murukannaiah. Platys Developers Guide. `http://research.csc.ncsu.edu/mas/platys/usage_dev.html`, 2012.

[81] Pradeep K. Murukannaiah, Nirav Ajmeri, and Munindar P. Singh. Acquiring creative requirements from the crowd: Understanding the influences of personality and creative potential in Crowd RE. In *Proceedings of the 24th IEEE International Requirements Engineering Conference*, pages 1–10 (To appear), Beijing, September 2016.

[82] Pradeep K. Murukannaiah, Nirav Ajmeri, and Munindar P. Singh. Engineering privacy in social applications. *IEEE Internet Computing*, 20(2):72–76, March 2016.

[83] Pradeep K. Murukannaiah, Ricard Fogues, and Munindar P. Singh. Platys: A framework for supporting context-aware personal agents. In *Proceedings of the 13th International Conference on Autonomous Agents and Multi-agent Systems*, pages 1689–1690, Paris, 2014.

[84] Pradeep K. Murukannaiah, Anup K. Kalia, Pankaj R. Telang, and Munindar P. Singh. Resolving goal conflicts via argumentation-based analysis of competing hypotheses. In *Proceedings of the 23rd IEEE International Requirements Engineering Conference*, pages 156–165, Ottawa, August 2015.

[85] Pradeep K. Murukannaiah and Munindar P. Singh. Platys Social: Relating shared places and private social circles. *IEEE Internet Computing*, 16(3):53–59, May 2012.

[86] Pradeep K. Murukannaiah and Munindar P. Singh. Understanding location-based user experience. *IEEE Internet Computing*, 18(6):72–76, November 2014.

[87] Pradeep K. Murukannaiah and Munindar P. Singh. Xipho: Extending Tropos to engineer context-aware personal agents. In *Proceedings of the 13th International Conference on Autonomous Agents and Multi-Agent Systems*, pages 309–316, Paris, 2014.

[88] Pradeep K. Murukannaiah and Munindar P. Singh. Platys: An active learning framework for place-aware application development and its evaluation. *ACM Transactions on Software Engineering and Methodology*, 24(3):1–33, May 2015.

[89] Gergely Palla, Imre Derényi, Illés J. Farkas, and Tamás Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814–818, April 2005.

[90] Gabriele Paolacci, Jesse Chandler, and Panagiotis G. Ipeirotis. Running experiments on Amazon Mechanical Turk. *Judgment and Decision Making*, 5(5):411–419, 2010.

[91] Elizabeth Paulos and Eric Goodman. The familiar stranger: Anxiety, comfort, and play in public places. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 223–230, April 2004.

[92] Eyal Peer, Joachim Vosgerau, and Alessandro Acquisti. Reputation as a sufficient condition for data quality on amazon mechanical turk. *Behavior Research Methods*, 46(4):1023–1031, 2014.

[93] Roger S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw Hill, New York, 6th edition, 2005.

[94] Iyad Rahwan, Thomas Juan, and Leon Sterling. Integrating social modelling and agent interaction through goal-oriented analysis. *Computer Systems Science and Engineering*, 21(2), 2006.

[95] Anand Ranganathan, Jalal Al-Muhtadi, Shiva Chetan, Roy Campbell, and M. Dennis Mickunas. MiddleWhere: A middleware for location awareness in ubiquitous computing applications. In *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware*, pages 397–416, Toronto, October 2004.

[96] Injong Rhee, Minsu Shin, Seongik Hong, Kyunghan Lee, and Song Chong. On the levy-walk nature of human mobility: Do humans walk like monkeys? In *Proceedings of 27th IEEE Conference on Computer Communications (INFOCOM)*, pages 924–932, April 2008.

[97] Maayan Roth, Asaf Ben-David, David Deutscher, Guy Flysher, Ilan Horn, Ari Leichtberg, Naty Leiser, Yossi Matias, and Ron Merom. Suggesting friends using the implicit social graph. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 233–242, Washington, DC, July 2010.

[98] Caspar Ryan and Atish Gonsalves. The effect of context and application type on mobile usability: An empirical study. In *Proceedings of the 28th Australasian Conference on Computer Science*, pages 115–124, Newcastle, Australia, February 2005.

[99] Mohammed Salifu, Yijun Yu, and Bashar Nuseibeh. Specifying monitoring and switching problems in context. In *Proceedings of the 15th IEEE International Requirements Engineering Conference*, pages 211–220, October 2007.

[100] Michele Sama, Sebastian Elbaum, Franco Raimondi, David S. Rosenblum, and Zhimin Wang. Context-aware adaptive applications: Fault patterns and their automated identification. *IEEE Transactions on Software Engineering*, 36(5):644–661, September 2010.

[101] Leila Scannell and Robert Gifford. Defining place attachment: A tripartite organizing framework. *Journal of Environmental Psychology*, 30(1):1–10, March 2010.

[102] Salvatore Scellato, Mirco Musolesi, Cecilia Mascolo, Vito Latora, and Andrew T. Campbell. NextPlace: A spatio-temporal prediction framework for pervasive systems. In *Proceedings of the 9th International Conference on Pervasive Computing*, pages 152–169, San Francisco, June 2011.

[103] Daniel Schuster, Alberto Rosi, Marco Mamei, Thomas Springer, Markus Endler, and Franco Zambonelli. Pervasive social context: Taxonomy and survey. *ACM Transactions on Intelligent Systems and Technology*, 4(3):1–22, July 2013.

[104] Estefanía Serral, Pedro Valderas, and Vicente Pelechano. Towards the model driven development of context-aware pervasive systems. *Pervasive and Mobile Computing*, 6(2):254–280, April 2010.

[105] Burr Settles. *Active Learning*. Morgan & Claypool, 2012.

[106] Quan Z. Sheng, Sam Pohlenz, Jian Yu, Hoi S. Wong, Anne H. H. Ngu, and Zakaria Maamar. ContextServ: A platform for rapid and flexible development of context-aware web services. In *Proceedings of the 31st International Conference on Software Engineering*, pages 619–622, Vancouver, 2009.

[107] Amit Sheth. Computing for human experience: Semantics-empowered sensors, services, and social computing on the ubiquitous web. *IEEE Internet Computing*, 14(1):88–91, January 2010.

[108] Munindar P. Singh. Self-renewing applications. *IEEE Internet Computing*, 15(4):3–5, July 2011.

[109] Manya Sleeper, Rebecca Balebako, Sauvik Das, Amber Lynn McConahy, Jason Wiese, and Lorrie Faith Cranor. The post that wasn't: Exploring self-censorship on Facebook. In *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW)*, pages 793–802, San Antonio, TX, 2013.

[110] Ian Smith, Sunny Consolvo, Anthony Lamarca, Jeffrey Hightower, James Scott, Timothy Sohn, Giovanni Iachello, and Gregory D. Abowd. Social disclosure of place: From location technology to communication practice. In *Proceedings of the 3rd International Conference on Pervasive Computing*, pages 134–151, Munich, May 2005.

[111] Derek J. Sollenberger and Munindar P. Singh. Kokomo: An empirically evaluated methodology for affective applications. In *Proceedings of the 10th International Conference on Autonomous Agents and MultiAgent Systems*, pages 293–300, Taipei, May 2011.

[112] Anna Cinzia Squicciarini, Mohamed Shehab, and Federica Paci. Collective privacy management in social networks. In *Proceedings of the 18th International Conference on World Wide Web (WWW)*, pages 521–530, Madrid, 2009.

[113] Graeme Stevenson, Juan Ye, Simon Dobson, and Paddy Nixon. LOC8: A location model and extensible framework for programming with location. *IEEE Pervasive Computing*, 9(1):28–37, January 2010.

[114] Jose M. Such and Natalia Criado. Adaptive conflict resolution mechanism for multi-party privacy management in social media. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, pages 69–72, 2014.

[115] Jose M. Such and Natalia Criado. Resolving multi-party privacy conflicts in social media. *arXiv:1507.04642*, 2015.

[116] Jose M. Such, Agustín Espinosa, and Ana García-Fornes. A survey of privacy in multi-agent systems. *Knowledge Engineering Review*, 29(03):314–344, 2014.

[117] Jose M. Such and Michael Rovatsos. Privacy policy negotiation in social media. *ACM Transactions on Autonomous and Adaptive Systems*, 2015. To appear.

[118] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Pearson, Boston, 2006.

[119] Kurt Thomas, Chris Grier, and David M. Nicol. unFriendly: Multi-party privacy risks in social networks. In *Proceedings of the 10th International Conference on Privacy Enhancing Technologies (PET)*, pages 236–252, Berlin, 2010.

[120] Mohit Tiwari, Prashanth Mohan, Andrew Osheroff, Hilfi Alkaff, Elaine Shi, Eric Love, Dawn Song, and Krste Asanović. Context-centric security. In *Proceedings of the 7th USENIX Conference on Hot Topics in Security*, pages 9–9, Bellevue, WA, 2012.

[121] Andreas Vlachos. Active learning with Support Vector Machines. Master's thesis, University of Edinburgh, 2004.

[122] Long Vu, Quang Do, and Klara Nahrstedt. Jyotish: Constructive approach for context predictions of people movement from joint Wifi/Bluetooth trace. *Pervasive and Mobile Computing*, 7(6):690–704, 2011.

[123] Douglas Walton, Christopher Reed, and Fabrizio Macagno. *Argumentation Schemes*. Cambridge University Press, 2008.

[124] Xinxi Wang, David Rosenblum, and Ye Wang. Context-aware mobile music recommendation for daily activities. In *Proceedings of the 20th ACM International Conference on Multimedia*, pages 99–108, Nara, Japan, 2012.

[125] Yang Wang, Gregory Norcie, Saranga Komanduri, Alessandro Acquisti, Pedro Giovanni Leon, and Lorrie Faith Cranor. I regretted the minute I pressed share: A qualitative study of regrets on Facebook. In *Proceedings of the Seventh Symposium on Usable Privacy and Security*, page 10, 2011.

[126] Jason Watson, Heather Richter Lipford, and Andrew Besmer. Mapping user preference to privacy default settings. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 22(6):32, 2015.

[127] Jason Wiese, Patrick Gage Kelley, Lorrie Faith Cranor, Laura Dabbish, Jason I. Hong, and John Zimmerman. Are you close with me? are you nearby?: Investigating social groups, closeness, and willingness to share. In *Proceedings of the 13th International Conference on Ubiquitous Computing (UbiComp)*, pages 197–206, Beijing, 2011.

[128] Matt Williams and Jon Williamson. Combining argumentation and Bayesian nets for breast cancer prognosis. *Journal of Logic, Language, and Information*, 15(1–2):155–178, 2006.

[129] Michael Winikoff and Lin Padgham. *Developing Intelligent Agent Systems: A Practical Guide*. Wiley, Chichester, UK, 2004.

[130] Ryan Wishart, Domenico Corapi, Srdjan Marinovic, and Morris Sloman. Collaborative privacy policy authoring in a social networking context. In *Proceedings of the IEEE International Symposium on Policies for Distributed Systems and Networks*, POLICY '10, pages 1–8, Fairfax, VA, July 2010.

[131] Pamela Wisniewski, Heather Richter Lipford, and David Wilson. Fighting for my space: Coping mechanisms for SNS boundary regulation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, pages 609–618, Austin, 2012.

[132] Juan Ye, Lorcan Coyle, Simon Dobson, and Paddy Nixon. A unified semantics space model. In *Proceedings of the 3rd International Conference on Location- and Context-Awareness*, pages 103–120, Oberpfaffenhofen, Germany, September 2007.

[133] Guangchao Yuan, Pradeep K. Murukannaiah, and Munindar P. Singh. Percimo: A personalized community model for location estimation in social media. In *Proceedings of the IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 1–8 (To appear), San Francisco, 2016.

[134] Guangchao Yuan, Pradeep K. Murukannaiah, Zhe Zhang, and Munindar P. Singh. Exploiting sentiment homophily for link prediction. In *Proceedings of the 8th ACM Conference on Recommender Systems*, pages 17–24, Foster City, CA, 2014.

[135] M. Zacarias, H. S. Pinto, R. Magalhães, and J. Tribolet. A 'context-aware' and agent-centric perspective for the alignment between individuals and organizations. *Information Systems*, 35(4):441–466, June 2010.

[136] Laura Zavala, Pradeep K. Murukannaiah, Nithyananthan Poosamani, Tim Finin, Anupam Joshi, Injong Rhee, and Munindar P. Singh. Platys: From position to place-oriented mobile computing. *AI Magazine*, 36(2):50–62, July 2015.

[137] Yu Zheng, Lizhu Zhang, Zhengxin Ma, Xing Xie, and Wei-Ying Ma. Recommending friends and locations based on individual location history. *ACM Transactions on the Web*, 5(1):1–29, February 2011.

[138] Changqing Zhou, Dan Frankowski, Pamela J. Ludford, Shashi Shekhar, and Loren G. Terveen. Discovering personally meaningful places: An interactive clustering approach. *ACM Transactions on Information Systems*, 25(3):1–31, July 2007.

[139] Xiaojin Zhu, Andrew B. Goldberg, Ronald Brachman, and Thomas Dietterich. *Introduction to Semi-Supervised Learning*. Morgan & Claypool, 2009.

[140] Zhenyun Zhuang, Kyu-Han Kim, and Jatinder Pal Singh. Improving energy efficiency of location sensing on smartphones. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, pages 315–330, San Francisco, 2010.

APPENDIX

Appendix A

Platys-Aware Application Development on Android

Platys currently supports place-aware application development on the Android platform (API level 10 and above). We briefly describe the steps involved in developing a Platys-aware Android application. First, a user installs the Platys middleware on an Android device and trains it to recognize place of his or her interest. Then, other applications on that device can interact with the middleware via interprocess communication (IPC) to acquire place information.

Platys middleware exposes an interface (Listing A.1) defined in Android Interface Definition Language (AIDL) declaring the functions a Platys-aware application can invoke [80]. Both the middleware and application must include a copy of the interface. The Android SDK Tools auto-generate an abstract implementation of the interface that acts as a stub (Listing A.2) on each end. The stub insulates developers from dealing with low-level details such as finding the remote process, and marshalling and unmarshalling data objects exchanged between the application and middleware. Further, the middleware defines a service (Listing A.3) that returns a concrete implementation of the stub when an application binds with the middleware.

Listing A.1: The Platys middleware exposes an inerface defined in AIDL to applications.

```
interface IPlatysMiddlewareRemoteService {
  /** Registers the application and returns a private key. */
  String registerApplication(String name, String description);

  void unregisterApplication(String privateKey);

  /** Status can be pending, approved, trashed, or blocked */
  String getAplicationStatus(String privateKey);

  /** Current place or null if not privileged */
  String getCurrentPlace(String privateKey);

  List<String> getCurrentActivities(String privateKey);

  List<String> getSocialCircles(String privateKey,
      String connectionName);

  List<String> getSharableSocialCircles(String privateKey,
      String placeOrActivityName)
  //...
}
```

Listing A.2: Android SDK Tools auto-generate a stub based on the AIDL interface exposed by the middleware.

```
public static abstract class Stub extends Binder implements
IPlatysMiddlewareRemoteService {
  /** Local-side IPC implementation stub class. */
```

```
  //...
}
```

Listing A.3: A service on the middleware returns a concrete implementation of the stub when an application binds with the middleware.

```
public class PlatysMiddlewareRemoteService extends Service {
  //...
  @Override
  public IBinder onBind(Intent intent) {
    return new IPlatysMiddlewareRemoteService.Stub() {
      /** A concrete implementation of
          IPlatysMiddlewareRemoteService */
      //...
    }
  }
}
```

Next, since the application also has a copy of the stub, it can bind to the middleware's service as if it is a local service (Listing A.4).

Listing A.4: An application can bind to the middleware's service and receive a concrete implementation of the stub.

```
private IPlatysMiddlewareRemoteService mService = null;
ServiceConnection mConnection = new ServiceConnection() {
  @Override
  public void onServiceConnected(ComponentName className,
      IBinder service) {
    mService = IPlatysMiddlewareRemoteService.Stub
        .asInterface(service);
```

```
    }
    @Override
    public void onServiceDisconnected(ComponentName className) {
      mService = null;
    }
};
```

Once bound, the application can interact with the middleware as follows.

1. Register with the middleware providing a unique name. The middleware returns a private the application should use in all future communication.

2. Check the status of the application; if approved, continue to next steps.

3. Query for the user's place, activity, or social circles as need be. The middleware returns the corresponding value or null according to the user's privacy policies.

4. The middleware sends asynchronous messages when the user's places or privacy policies change so that the application can update local caches without polling.

5. Unregister the application if place information is not needed anymore.