Platys: An Active Learning Framework for Place-Aware Application Development and Its Evaluation

Pradeep K. Murukannaiah Munindar P. Singh

Abstract

We introduce a high-level abstraction of location called *place*. A place is not necessarily centered on physical space, and derives its meaning from a user's activities and social context. In this manner, place can facilitate improved user experience compared to traditional representation of location, which is spatial coordinates. We propose *Platys* framework as a way to address the special challenges of place-aware application development. The crux of Platys is a middleware that 1. learns a model of places specific to each user via *active learning*, a machine learning paradigm, to reduce the user-effort required for training the middleware, and 2. exposes the learned user-specific model of places to applications at run time, insulating application developers from dealing with low-level sensors and nuances in how users perceive places.

We evaluated Platys via two studies. First, we applied Platys' active learning approach to learn a model of places for each of 10 users from the user's place labels and sensor data collected from an Android phone. Compared with two supervised and two unsupervised place learning approaches, we found that Platys 1. requires fewer place labels than traditional supervised approaches to learn a user's places with desired accuracy, and 2. learns places with higher accuracy than unsupervised approaches.

Second, we conducted a developer study to evaluate Platys' efficiency in assisting developers and its effectiveness in enabling usable applications. In this study, 46 developers employed Platys or the Android location API to develop a place-aware application. Our results indicate that application developers employing Platys, when compared to those employing the Android API, 1. develop a location-aware application faster and perceive reduced difficulty and 2. produce applications that are easier to understand (for developers) and potentially more usable and privacy preserving (for application users).

1 Introduction

Location awareness is an important feature of mobile applications including search, social networking, and games. In many cases, a user would not even notice that an application is location-aware. As Weiser [1999] observed, “the most profound technologies are those that disappear.”

Weiser’s vision leads to three major questions that we seek to address in this paper.

1. What are the levels of abstraction (or granularity) at which a mobile application could employ the location information?

2. What are the implications of the chosen level of abstraction on the process of location-aware application development?

3. How does the chosen level of location abstraction affect the quality of the applications produced both from the perspectives of application developers and end-users?

In current practice, most location-aware applications represent location 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 Küpper [2005]. 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 an application developer.

1.1 From Position to Place

Applications employ location in some typical ways:

1. *Explicit.* Using the information as is, as below:
    - *Informative.* An application can provide a user’s location information explicitly, e.g., it may display the location in a calendar or tag location on a photo.
    - *Social disclosure.* An application can disclose a user’s location to the social contacts of the user, e.g., on a social network site or in a text message.
    - *Commercial disclosure.* An application can disclose a user’s location to a third party for a commercial purpose, e.g., to obtain a coupon for a nearby coffee shop.

2. *Implicit.* An application can automate a task based on the user’s location. For example, consider a personalization task such as changing the ringer mode of the phone or forwarding text messages to email, which can be performed automatically depending upon where the user is.

3. *Prediction.* An application 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 application employs location for the user to benefit from the location personally (for information or task execution), or exploits location 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 *place*. Employing place instead of position has three implications on location-aware applications.

- By presenting location in a way that is natural to users, place can enhance *usability*, i.e., the ease with which a user can exercise an application Ryan and Gonsalves [2005].
- Place opens up new avenues for intelligent location-aware applications including social networks Murukannaiah and Singh [2012], personal assistants desJardins et al. [2005], pervasive and social games Magerkurth et al. [2005], recommender systems Wang et al. [2012], and virtual worlds Hendaoui et al. [2008].
- Place can enhance *location privacy* Duckham and Kulik [2006] 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.

1.2 Place-Aware Application Development

Although place offers potential benefits as a location abstraction, place-aware application development is quite challenging.

First, how can a place-aware application represent and reason about the places that 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.

Second, 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.

Third, we need to provide architectural support for developing a place-aware application including means for modeling and acquiring place information.

1.3 Contributions and Organization

We make two main contributions. First, we propose the Platys framework for place-aware application development. The framework incorporates a *middleware* providing the architectural support necessary for place-aware application development (Section 3). Before describing the middleware, we synthesize a conceptual *metamodel* based on various place-related constructs (Section 2). Second, we describe Platys Reasoner, a key component of the middleware, which reasons about places from sensor data (Section 4). Platys Reasoner is novel in that it 1. reasons about a user's places subjectively, enhancing the user experience delivered by place-aware applications, 2. prompts the user to label places only if required, reducing the user effort involved in training the reasoner, and 3. makes realistic assumptions that sensors readings for place recognition will be from multiple sources and intermittent.

We evaluate Platys via two studies involving real users.

1. A user study finds that Platys Reasoner is effective for place recognition, reducing user effort and enhancing place recognition accuracy (Section 5).

2. A developer study finds that Platys framework is effective for place-aware application development, reducing development time and effort, and potentially enhancing the usability of location-aware applications (Section 6).

We summarize the related work in Section 7, identify some directions for future research in Section 8, and conclude in Section 9.

2 A Conceptual Metamodel of Place

The notion of place has been studied under constructs such as place attachment, place identity, sense of place, and semantic location. Gieryn [2000] identifies geographic location, material form, and meaning and value as three features of a place. Scannell and Gifford [2010] 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) Goel et al. [2011], Kyle and Chick [2007], Milligan [1998]. Harrison and Dourish [1996] distinguish space and place by a phrase that "we are located in space, but we act in place."

We synthesize (as Lewicka [2011] advocates) various place-related constructs as an informational entity that can be computed and employed in mobile applications. Figure 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.

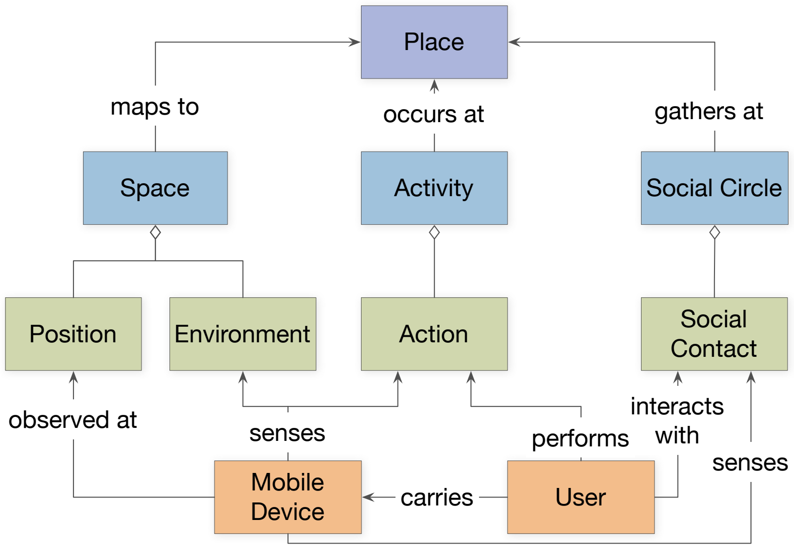


Figure 1: A conceptual metamodel relating place to space, activities, and social circles.

- *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.

- *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 Murukannaiah and Singh [2012]. 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.

The notion of *context* is related to place. In contrast to context, which is defined as "any relevant information" Dey et al. [2001], 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. This helps 1. developers determine if Platys provides the abstractions they desire, and 2. end-users train Platys appropriately (we imagine that asking a user about his or her space, activities, or social circles is clearer than asking about context).

3 Platys: Framework for Place-Aware Application Development

As shown in Figure 2, Platys framework 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.

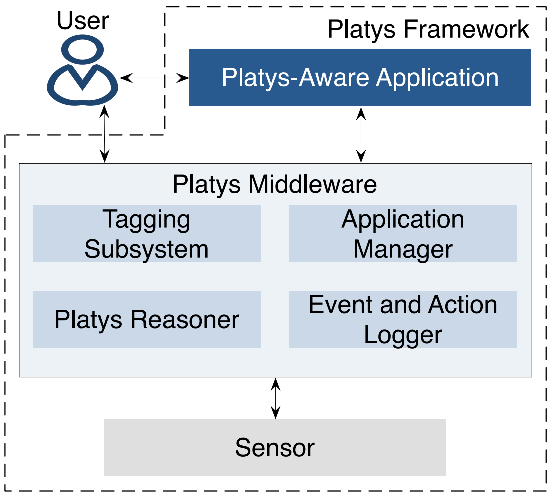


Figure 2: Platys framework consists of a middleware, sensors, and applications.

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 Emmerich [2000], Issarny et al. [2007] 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 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.

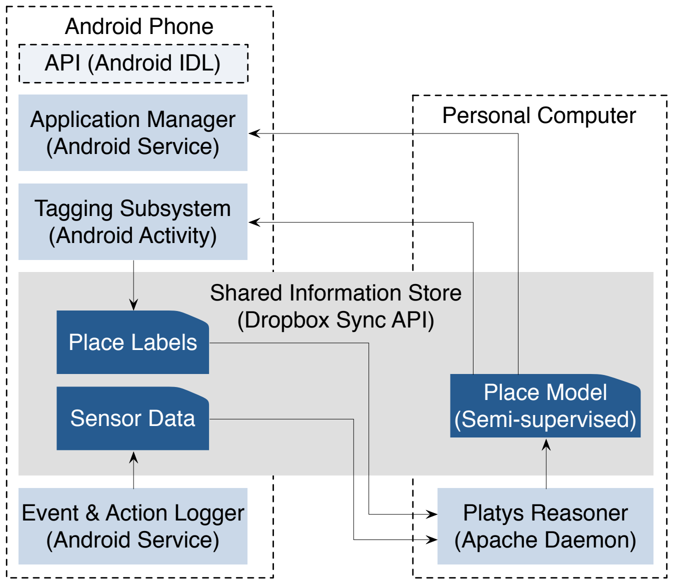

Android Phone
API (Android IDL)
Application Manager (Android Service)
Tagging Subsystem (Android Activity)
Personal Computer
Shared Information Store (Dropbox Sync API)
Place Labels
Sensor Data
Place Model (Semi-supervised)
Event & Action Logger (Android Service)
Platys Reasoner (Apache Daemon)

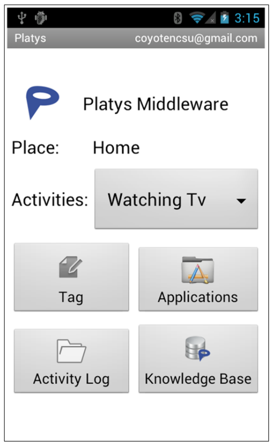
Figure 3: Platys middleware's subsystems. The subsystems are loosely coupled and communicate asynchronously via a shared information store. Each of a user's personal devices can host one or more of the subsystems.

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.

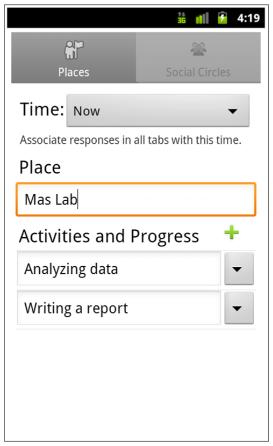
For Platys to make sense of the sensor data, the tagging subsystem helps a user train Platys on the relevant place, activity, and social circle. Since smart phones are always with a user, they are ideal to deliver notifications prompting the user to tag. Figure 4b shows the user interface from our Android implementation of the tagging subsystem. The user may ignore any prompt 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 (Section 4). 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 4c and 4d. An instance of the application manager must be hosted on each device which hosts place-aware applications.

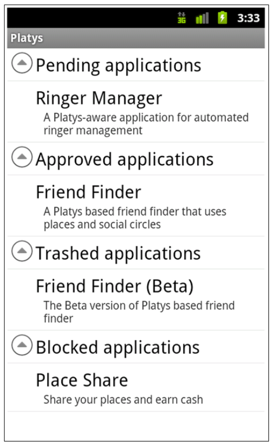

3:15
Platys coyotencsu@gmail.com
Platys Middleware
Place: Home
Activities: Watching Tv

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

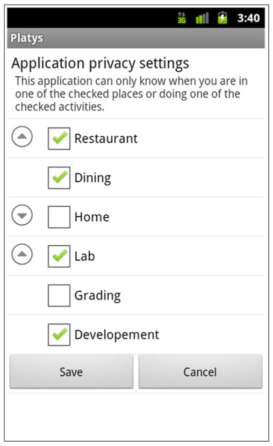

4:19

Time: Now
Associate responses in all tabs with this time.
Place
Mas Lab
Activities and Progress
Analyzing data
Writing a report

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


3:33
Platys
Pending applications
Ringer Manager
A Platys-aware application for automated ringer management
Approved applications
Friend Finder
A Platys based friend finder that uses places and social circles
Trashed applications
Friend Finder (Beta)
The Beta version of Platys based friend finder
Blocked applications
Place Share
Share your places and earn cash

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


3:40
Platys
Application privacy settings
This application can only know when you are in one of the checked places or doing one of the checked activities.
- [x] Restaurant
- [x] Dining
- [ ] Home
- [x] Lab
- [ ] Grading
- [x] Developement

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

Figure 4: Screenshots from the Platys middleware's subsystems.

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 5. Platys-aware application development tackles the challenges in discussed in Section 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. Code snippets of how an application interacts with the middleware are shown in Appendix B.

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 Smith et al. [2005]. Platys supports such fine-grained privacy policies.

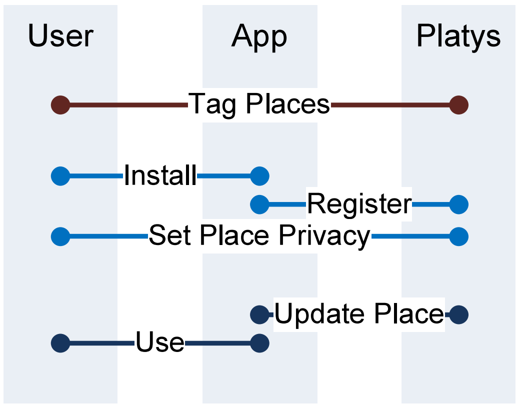


Figure 5: Interactions between a user, a Platys-aware application, and the Platys middleware. The user tags places on an ongoing basis; the application is installed and registered; the middleware continually sends place updates to the application as needed.

4 Platys Reasoner

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* Hariharan and Toyama [2004], Montoliu and Gatica-Perez [2010], Zheng et al. [2011]. 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 Lin et al. [2010]. Three, staypoint-based approaches, often, require frequent sensor readings to find patterns in the unlabeled data. For example, Ashbrook and Starner [2002] and Zhou et al. [2007] 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) Hastie et al. [2001] 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).

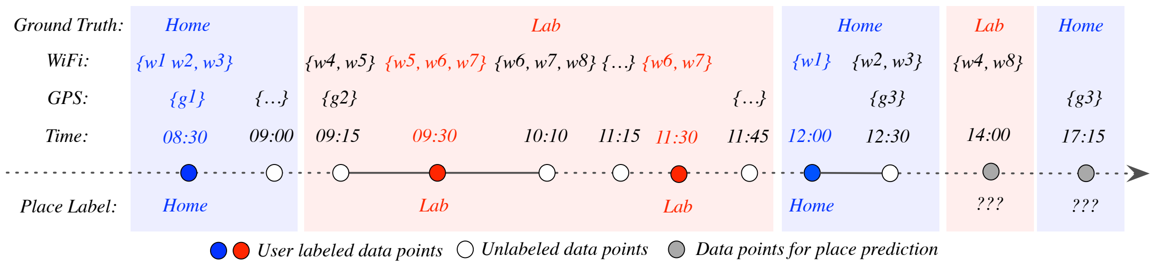
The Platys Reasoner seeks to address the shortcomings of the above approaches. Specifically, it combines two machine learning paradigms: (1) *active learning* Settles [2012] to reduce the labeling effort, and (2) *semi-supervised learning* Zhu et al. [2009] to efficiently deal with intermittent and infrequent sensor data. Next, we explain these paradigms and how Platys Reasoner employs them for place recognition. We provide a formal description of the problem and solution (with pseudocode) in Appendix A.

4.1 Intuition: Active and Semi-supervised Learning

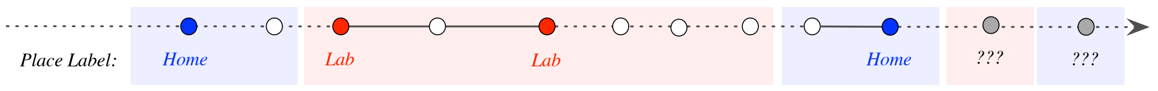
As an example, Alex is a Platys user. Figure 6a captures Alex's routine on a typical day: Alex is at *home* in the morning, during lunch, and evening; he works from his *lab* during a morning session and an afternoon session. As shown, Alex has labeled the two places, twice each, and there are sporadic 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 $W(\text{14:00}) = \{w^4, w^8\}$, what is Alex's place at 14:00? Similarly, given historical data and $G(\text{17:15}) = \{g^3\}$, what is his place at 17:15?

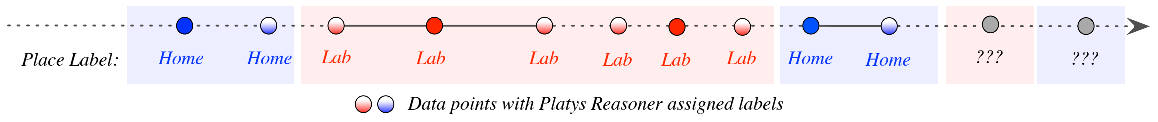
In general, an unsupervised approach cannot answer the above question because it does not employ place labels. In contrast, a traditional supervised approach may not be accurate



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

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 $PL(8{:}30) =$ *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 6b. 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 $PL(9{:}30) =$ *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 $PL(9{:}15) =$ *lab* and $PL(10{:}10) =$ *lab* as shown in Figure 6c, and then train a classifier. Such a classifier would predict $PL(14{:}00)$ as *lab* correctly, enhancing the classification accuracy.

4.2 Place Recognition Pipeline

Platys Reasoner incorporates the above intuitions by employing active and semi-supervised learning techniques, as shown in Figure 7. 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.

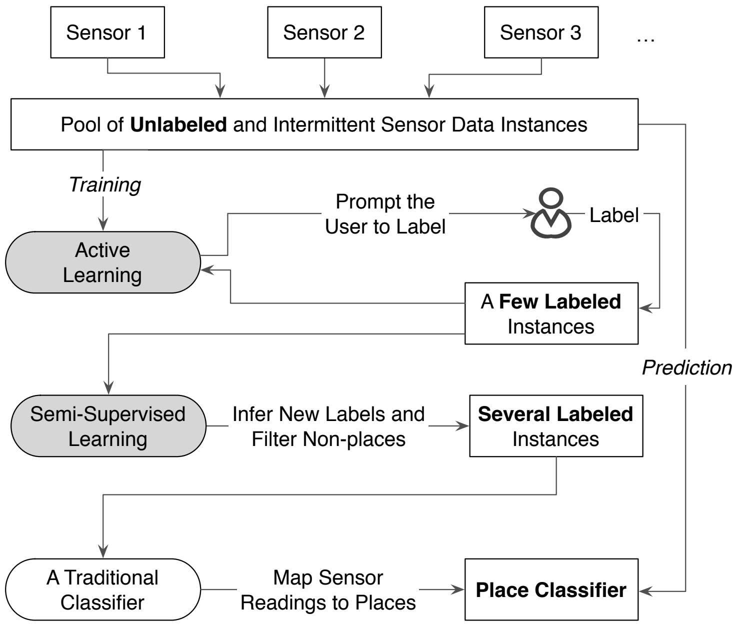


Figure 7: Platys Reasoner learns a place classifier from unlabeled sensor data.

4.2.1 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* Settles [2012]:

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 6). 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 Vlachos [2004].

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.

4.2.2 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 Eagle and Pentland [2006]. We employ this intuition in a semi-supervised technique called *self training* Zhu et al. [2009]. 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 A.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.

4.3 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 Azizyan et al. [2009]) and the activity component (e.g., via accelerometer Kwapisz et al. [2011]). However, how do we recognize the third component of our place metamodel—social circles?

Traditionally, *community detection* Fortunato [2010] 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 Murukannaiah and Singh [2012] 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 Palla et al. [2005] 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.

5 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.

5.1 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 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.

5.2 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.

Table 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

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 Bishop [2006] 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.

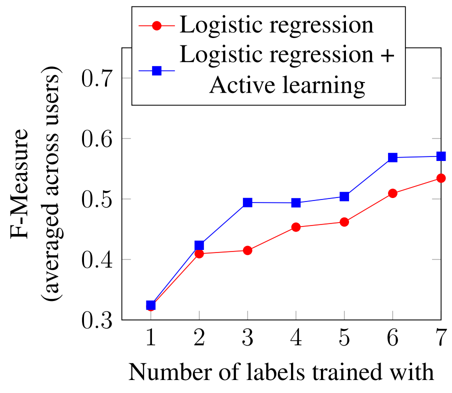
5.3 Comparison with Two Traditional Classifiers (Supervised)

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.

5.3.1 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 8 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).



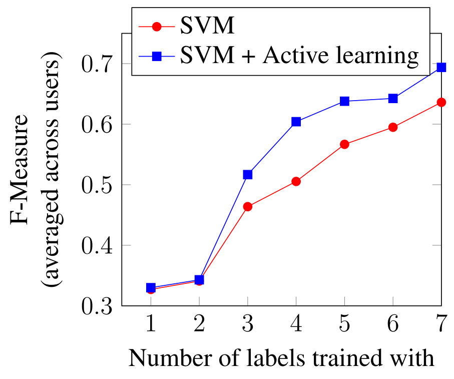
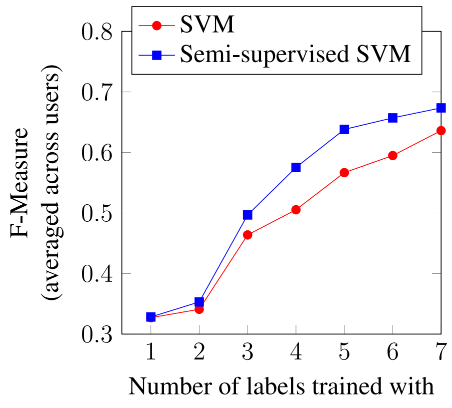


Figure 8: Platys Reasoner's active learner compared with two traditional classifiers.

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.

5.3.2 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 9 (left), the semi-supervised SVM achieves a higher F-measure than SVM.



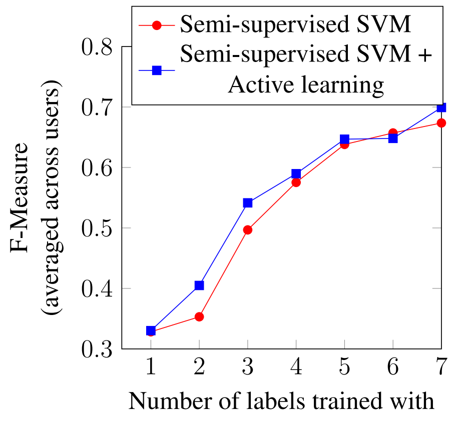


Figure 9: Platys Reasoner's semi-supervised and active learning compared for SVM.

Our place-recognition pipeline employs both semi-supervised and active learning techniques. Figure 9 (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.

5.4 Comparison with Two Staypoint-Based Approaches (Unsupervised)

We compared Platys Reasoner with two staypoint-based approaches Hariharan and Toyama [2004], Zheng et al. [2011]. 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 $\mathrm{m}\rangle$ to $\langle$1 day, 96 $\mathrm{km}\rangle$.

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

(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 $\langle$30 minute, 200 $\mathrm{m}\rangle$ used by Zheng et al. [2011] 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.

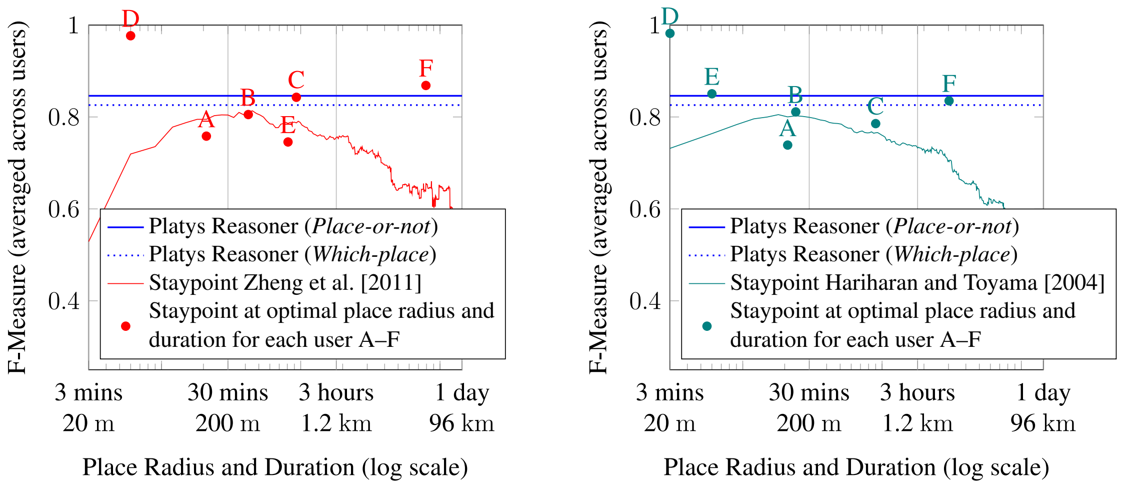


Figure 10: Platys Reasoner compared with two staypoint-based approaches.

6 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.

6.1 Study Design

Our study design terminology is adapted from Juristo and Moreno [2001]. For convenience, we summarize the key terms (with their mapping to our study) in Table 2.

6.1.1 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.

Table 2: 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.

- 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.

6.1.2 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.

6.1.3 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.

6.1.4 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.

6.1.5 Response Variables

Table 3 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 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 McCabe [1976], Pressman [2005] 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.

Table 3: 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**
$time_{subtask}$ $time_{RMS}$ $time_{RMS-prep}$	$subtask$ RMS RMS after $prep$	Development time reported by subjects.
$effort_{subtask}$ $effort_{RMS}$ $effort_{RMS-prep}$	$subtask$ RMS RMS after $prep$	Perceived effort as reported by subjects.
MCC $NoLM$ NoS NoM	RMS	McCabe's cyclomatic complexity. Number of levels per method. Number of statements. Number of methods (for a fixed NoS).
$usability$	RMS	Extent to which an RMS implementation meets usability and privacy requirements.

- $NoLM$: 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.

6.1.6 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 Android Open Source Project [2012] for location acquisition, which provides position information from GPS or the network. The *Platys Group* employed the Platys middleware as Murukannaiah [2012] 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 7), none are available for easy deployment on the Android platform to enable a fair comparison with Platys.

6.1.7 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 1. strategies for enhancing usability and privacy of RMS, and 2. 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.

6.2 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 4. 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 4: 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 Freund and Perles [2004], Hollander and Wolfe [1999].

6.3 Results and Discussion: Time and Effort

Figures 11 and 12 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).

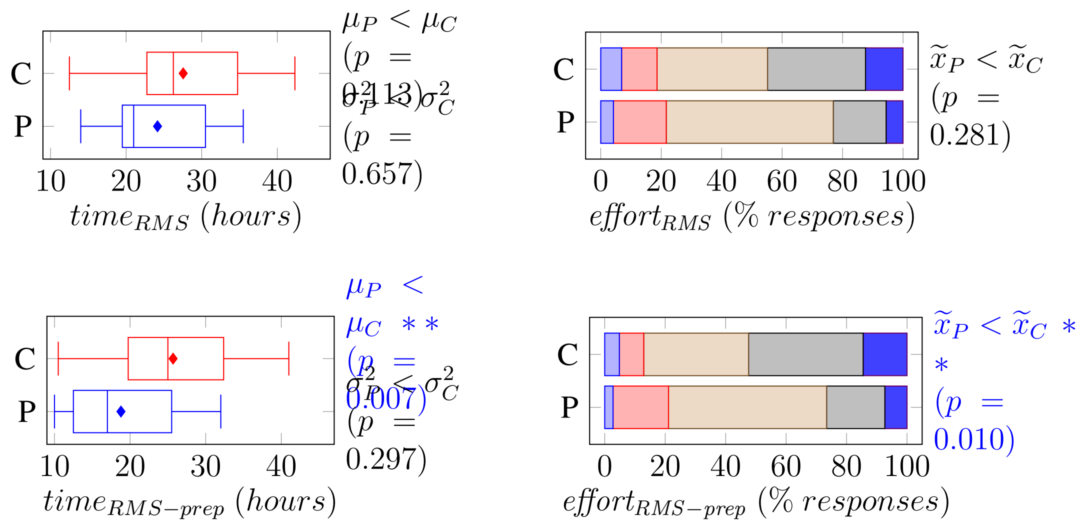


Figure 11: Comparison of the time (left) and effort expended (right) by Platys (P) and Control (C) subjects to develop RMS, highlighting significant differences.

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.

6.3.1 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 developer. 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.

6.3.2 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

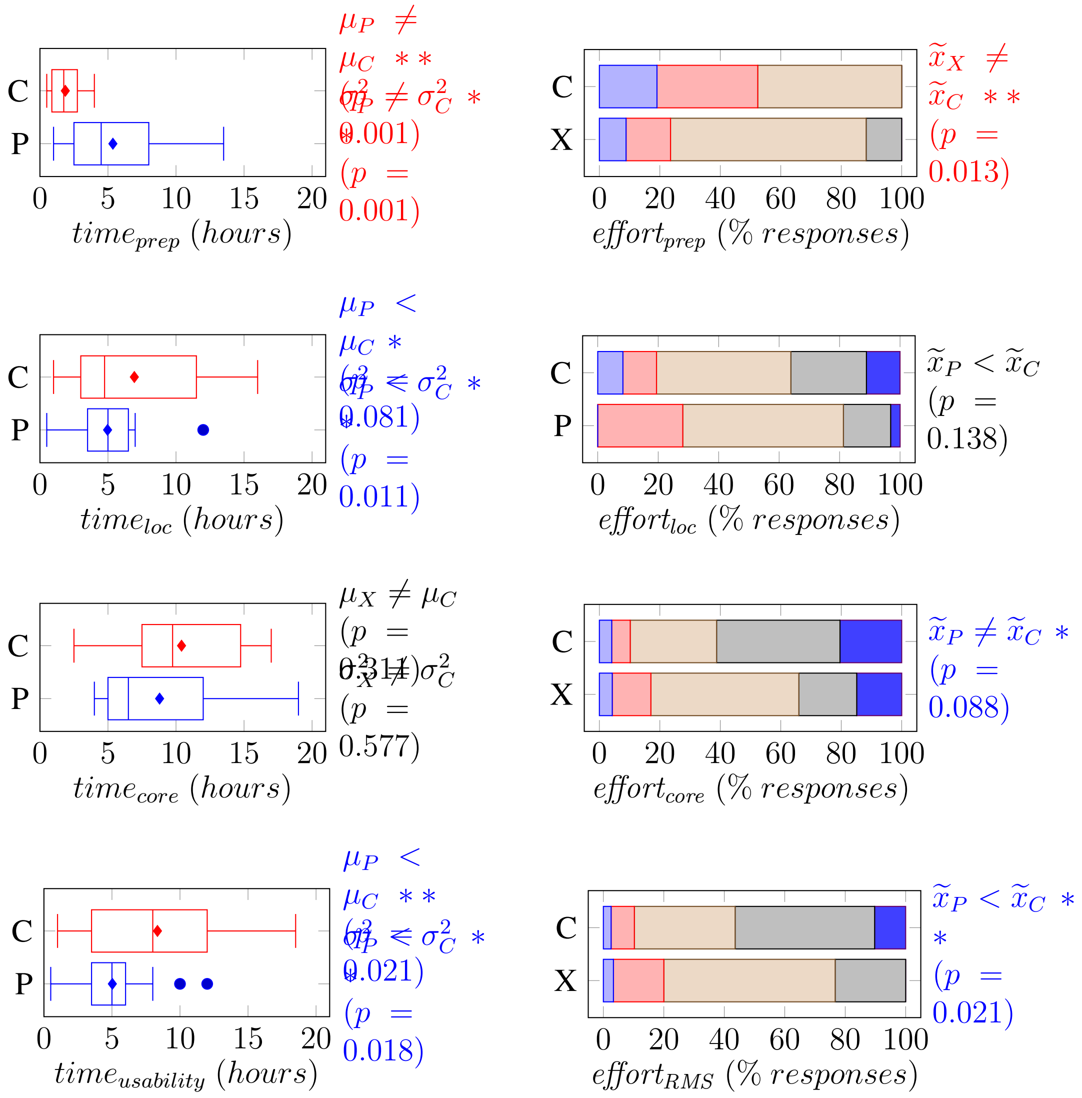


Figure 12: Comparison of the time (left) and effort expended (right) by Platys (P) and Control (C) subjects to develop RMS' subtasks, highlighting significant differences.

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.

6.3.3 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.

6.3.4 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%.

6.3.5 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.

6.4 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 13 compare the software metrics of the RMS applications developed by Control and Platys subjects (computed from the source code). The figure also

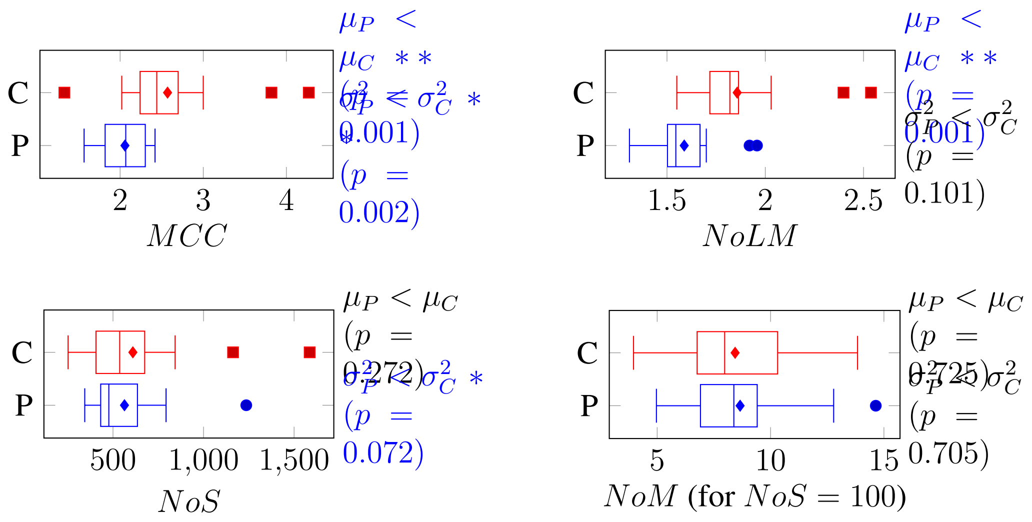


Figure 13: Comparison of the software code metrics for the RMS implementations produced by the Platys (P) and Control (C) subjects. Also shown are the results of hypothesis testing, highlighting significant differences (to the right of each plot).

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 13) 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 the mean code size of Control implementations would be higher if all Control implementations met all usability requirements (Section 6.5).

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.

6.5 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 5 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 5: 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

6.5.1 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*.

6.5.2 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.

6.5.3 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.

7 Related Work

Existing works on acquiring places (or a similar construct) from low-level sensors, and providing engineering support for developing place-aware applications typically lie at different ends of a spectrum. Platys seamlessly integrates the two providing an end-to-end solution, as well as offering distinct advantages. Below, we identify related works in both areas.

7.1 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 2004 MiddleWhere middleware realizes a hierarchical model of location involving points, lines, and polygons backed by physical coordinates. Stevenson et al.'s 2010 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. [2007] describe additional systems that realize space models. Approaches that model only space and spatial relationships fail to capture place in its entirety.

Baldauf et al. [2007] survey several context-aware systems. Bettini et al. [2010] provide a comprehensive view of several context modeling and reasoning techniques. Although existing context-aware systems model more than space, they largely focus on environmental features of context. Schuster et al. [2013] survey several systems that bringing together spatio-temporal aspects of context and online user interactions (e.g., on a social network site). These systems 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. Also, Platys promotes privacy by running locally on a user's personal devices.

7.2 Place Acquisition

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 Starner [2002] 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. [2007] learn places from one-minute frequent GPS logs, though via density-based clustering, and obtain better accuracy than the k-means approach. NextPlace Scellato et al. [2011] models the significance of a GPS coordinate as a Gaussian distribution based on the length of a user's stay at the coordinate and at the coordinates next to it. NextPlace considers, as places, only those coordinates that have a significance higher than a specified threshold. Similarly, Zheng et al. [2011] and Hariharan and Toyama [2004] extract staypoints via clustering (with fixed staypoint parameters $\langle$30 minute, 200 $\mathrm{m}\rangle$) and probabilistic approaches, respectively.

Unlike the GPS-based approaches above, Kang et al. [2005] 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. [2011] 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. [2005] 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 Kim et al. [2010] 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.

7.3 Place-Aware Application Development

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

CARISMA Capra et al. [2003] 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.

Topiary Li et al. [2004] 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 application of the Platys middleware so that the tool could receive context components from Platys (unlike the fixed, built-in, context components of Topiary). Topiary is informally evaluated for understandability, ease-of-use, and usefulness.

LIME (Linda in a Mobile Environment) Murphy et al. [2006] 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) Mamei and Zambonelli [2009] 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 Guo et al. [2011] 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 Buthpitiya et al. [2012] 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, Platys middleware implements the place reasoner as one widget rather than several context widgets that Hermes employs. We conjecture that a unified treatment of place enhances the intelligibility and simplifies design.

Kulkarni et al. [2012] 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 the developers and end-users.

To the best of our knowledge, the Platys developer study 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 understand 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.

8 Directions

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

8.1 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 involves and the battery power it consumes. Platys addresses the first 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 Zhuang et al. [2010]. 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. [2012] 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.

8.2 Usability Evaluation of Place-Aware Applications

We performed a qualitative evaluation of the usability of RMS applications. However, Duh et al. [2006] 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 [2005] conducted a field study and found that location (position) can significantly affect the usability of a mobile application. But how different abstractions of location such as position and place can 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.

8.3 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. [2007] 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 Ali et al. [2013] during requirements phase is valuable in that inconsistencies and conflicts in location-based requirements can be detected early. However, we understand that specifying a complete set of place-based requirements during design is difficult since places of interest to a user are often unknown a priori and are subject to evolution. In such circumstances, automated discovery, at run time, of fault *patterns* Sama et al. [2010] could be a viable option. We will explore the possibility of incorporating such options in the Platys middleware in future.

9 Conclusion

Intelligent location-aware applications are being widely adopted. Yet, these applications are often developed in an ad hoc manner and yield 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. The Platys-aware application development platform addresses the challenges of providing a communication channel between the users of a location-aware mobile application and its developers, providing a uniform place model across applications, and providing 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 extra 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

We thank the National Science Foundation for partial support under grant 0910868 and Chung-Wei Hang for useful discussions.

References

ALI, R., DALPIAZ, F., AND GIORGINI, P. 2013. Reasoning with contextual requirements: Detecting inconsistency and conflicts. *Information and Software Technology 55,* 1, 35–57.

ANDROID OPEN SOURCE PROJECT. 2012. Android Developers Guide: Obtaining user location. `http://developer.android.com/guide/topics/location/obtaining-user-location.html`.

ASHBROOK, D. AND STARNER, T. 2002. Learning significant locations and predicting user movement with GPS. In *Proceedings of the 6th International Symposium on Wearable Computers (ISWC)*. IEEE Computer Society, Seattle, WA, 101–108.

AZIZYAN, M., CONSTANDACHE, I., AND ROY CHOUDHURY, R. 2009. SurroundSense: Mobile phone localization via ambience fingerprinting. In *Proceedings of the 15th Annual International Conference on Mobile Computing and Networking*. ACM, New York, 261–272.

BALDAUF, M., DUSTDAR, S., AND ROSENBERG, F. 2007. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing 2,* 4, 263–277.

BETTINI, C., BRDICZKA, O., HENRICKSEN, K., INDULSKA, J., NICKLAS, D., RANGANATHAN, A., AND RIBONI, D. 2010. A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing 6,* 2, 161–180.

BISHOP, C. M. 2006. *Pattern Recognition and Machine Learning*. Springer, Secaucus, NJ.

BUTHPITIYA, S., LUQMAN, F., GRISS, M., XING, B., AND DEY, A. K. 2012. 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*. IEEE Computer Society, Washington, DC, 663–670.

CAPRA, L., EMMERICH, W., AND MASCOLO, C. 2003. CARISMA: Context-aware reflective middleware system for mobile applications. *IEEE Transactions on Software Engineering 29,* 10, 929–945.

DESJARDINS, M., EATON, E., AND WAGSTAFF, K. 2005. A context-sensitive and user-centric approach to developing personal assistants. In *Working Notes of the AAAI Spring Symposium on Persistent Assistants*. AAAI, 98–100.

DEY, A. K., ABOWD, G. D., AND SALBER, D. 2001. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction 16,* 2, 97–166.

DUCKHAM, M. AND KULIK, L. 2006. Location privacy and location-aware computing. In *Dynamic and Mobile GIS: Investigating Changes in Space and Time*, J. Drummond, R. Billen, E. João, and D. Forrest, Eds. CRC Press, Boca Raton, FL, Chapter 3, 34–51.

DUH, H. B.-L., TAN, G. C. B., AND CHEN, V. H.-H. 2006. 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*. ACM, New York, 181–186.

EAGLE, N. AND PENTLAND, A. S. 2006. Reality mining: Sensing complex social systems. *Personal and Ubiquitous Computing 10,* 4, 255–268.

EMMERICH, W. 2000. Software engineering and middleware: A roadmap. In *Proceedings of the Conference on the Future of Software Engineering*. ACM, New York, 117–129.

FORTUNATO, S. 2010. Community detection in graphs. *Physics Reports 486,* 3–5, 75–174.

FREUND, J. E. AND PERLES, B. M. 2004. *Statistics: A First Course*. Prentice Hall, Upper Saddle River, NJ.

GIERYN, T. F. 2000. A space for place in sociology. *Annual Review of Sociology 26,* 1, 463–496.

GOEL, L., JOHNSON, N. A., JUNGLAS, I., AND IVES, B. 2011. From space to place: Predicting users' intentions to return to virtual worlds. *Management Information Systems Quarterly 35,* 3, 749–771.

GOOGLE. 2013. Google Places API. `https://developers.google.com/places/`.

GUO, B., ZHANG, D., AND IMAI, M. 2011. Toward a cooperative programming framework for context-aware applications. *Personal and Ubiquitous Computing 15,* 3, 221–233.

HARIHARAN, R. AND TOYAMA, K. 2004. Project Lachesis: Parsing and modeling location histories. In *Proceedings of the 3rd International Conference on Geographic Information Science*. Springer, 106–124.

HARRISON, S. AND DOURISH, P. 1996. 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*. ACM, New York, 67–76.

HASTIE, T., TIBSHIRANI, R., AND FRIEDMAN, J. 2001. *The Elements of Statistical Learning*. Springer, New York.

HENDAOUI, A., LIMAYEM, M., AND THOMPSON, C. W. 2008. 3D social virtual worlds: Research issues and challenges. *IEEE Internet Computing 12,* 1, 88–92.

HIGHTOWER, J., CONSOLVO, S., LAMARCA, A., SMITH, I. E., AND HUGHES, J. 2005. Learning and recognizing the places we go. In *Proceedings of the 7th International Conference on Ubiquitous Computing*. Springer, 159–176.

HOLLANDER, M. AND WOLFE, D. A. 1999. *Nonparametric Statistical Methods*. Wiley, New York.

ISSARNY, V., CAPORUSCIO, M., AND GEORGANTAS, N. 2007. A perspective on the future of middleware-based software engineering. In *Proceedings of the Conference on the Future of Software Engineering*. IEEE Computer Society, Washington, DC, 244–258.

JURISTO, N. AND MORENO, A. M. 2001. *Basics of Software Engineering Experimentation*. Kluwer Academic Publishers, Boston/Dordrecht/London.

KANG, J. H., WELBOURNE, W., STEWART, B., AND BORRIELLO, G. 2005. Extracting places from traces of locations. *SIGMOBILE Mobile Computing Communications Review 9,* 3, 58–68.

KIM, D. H., KIM, Y., ESTRIN, D., AND SRIVASTAVA, M. B. 2010. SensLoc: sensing everyday places and paths using less energy. In *Proceedings of the 8th Conference on Embedded Networked Sensor Systems*. ACM, 43–56.

KULKARNI, D., AHMED, T., AND TRIPATHI, A. 2012. A generative programming framework for context-aware CSCW applications. *ACM Transactions on Software Engineering and Methodology 21,* 2, 1–35.

KÜPPER, A. 2005. *Location-Based Services: Fundamentals and Operation*. John Wiley and Sons, Chichester, UK.

KWAPISZ, J. R., WEISS, G. M., AND MOORE, S. A. 2011. Activity recognition using cell phone accelerometers. *SIGKDD Explorations 12,* 2, 74–82.

KYLE, G. AND CHICK, G. 2007. The social construction of a sense of place. *Leisure Sciences 29,* 3, 209–225.

LEWICKA, M. 2011. Place attachment: How far have we come in the last 40 years? *Journal of Environmental Psychology 31,* 3, 207–230.

LI, Y., HONG, J. I., AND LANDAY, J. A. 2004. Topiary: A tool for prototyping location-enhanced applications. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*. ACM, New York, 217–226.

LIN, D. 1998. An information-theoretic definition of similarity. In *Proceedings of the 15th International Conference on Machine Learning*. Morgan Kaufmann, Madison, WI, 296–304.

LIN, J., XIANG, G., HONG, J. I., AND SADEH, N. M. 2010. Modeling people's place naming preferences in location sharing. In *Proceedings of the 12th International Conference on Ubiquitous Computing (UbiComp)*. ACM Press, Copenhagen, Denmark, 75–84.

MAGERKURTH, C., CHEOK, A. D., MANDRYK, R. L., AND NILSEN, T. 2005. Pervasive games: Bringing computer entertainment back to the real world. *Computers in Entertainment 3,* 3, 4–4.

MAMEI, M. AND ZAMBONELLI, F. 2009. Programming pervasive and mobile computing applications: The TOTA approach. *ACM Transactions on Software Engineering and Methodology 18,* 4, 1–56.

MCCABE, T. J. 1976. A complexity measure. *IEEE Transactions on Software Engineering 2*, 308–320.

MILLIGAN, M. J. 1998. Interactional past and potential: The social construction of place attachment. *Symbolic Interaction 21,* 1, 1–33.

MONTOLIU, R. AND GATICA-PEREZ, D. 2010. Discovering human places of interest from multimodal mobile phone data. In *Proceedings of the 9th International Conference on Mobile and Ubiquitous Multimedia*. ACM Press, 12:1–12:10.

MURPHY, A. L., PICCO, G. P., AND ROMAN, G.-C. 2006. LIME: A coordination model and middleware supporting mobility of hosts and agents. *ACM Transactions on Software Engineering and Methodology 15,* 3, 279–328.

MURUKANNAIAH, P. K. 2012. Platys Developers Guide. `http://research.csc.ncsu.edu/mas/platys/usage_dev.html`.

MURUKANNAIAH, P. K. AND SINGH, M. P. 2012. Platys Social: Relating shared places and private social circles. *IEEE Internet Computing 16,* 3, 53–59.

PALLA, G., DERÉNYI, I., FARKAS, I. J., AND VICSEK, T. 2005. Uncovering the overlapping community structure of complex networks in nature and society. *Nature 435,* 7043, 814–818.

PRESSMAN, R. S. 2005. *Software Engineering: A Practitioner's Approach* 6th Ed. McGraw Hill, New York.

RANGANATHAN, A., AL-MUHTADI, J., CHETAN, S., CAMPBELL, R., AND MICKUNAS, M. D. 2004. MiddleWhere: A middleware for location awareness in ubiquitous computing applications. In *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware*. Springer-Verlag, New York, 397–416.

RYAN, C. AND GONSALVES, A. 2005. The effect of context and application type on mobile usability: An empirical study. In *Proceedings of the 28th Australasian Conference on Computer Science*. Australian Computer Society, Inc., Darlinghurst, Australia, 115–124.

SALIFU, M., YU, Y., AND NUSEIBEH, B. 2007. Specifying monitoring and switching problems in context. In *Proceedings of the 15th IEEE International Requirements Engineering Conference*. IEEE Computer Society, Los Alamitos, CA, 211–220.

SAMA, M., ELBAUM, S., RAIMONDI, F., ROSENBLUM, D. S., AND WANG, Z. 2010. Context-aware adaptive applications: Fault patterns and their automated identification. *IEEE Transactions on Software Engineering 36,* 5, 644–661.

SCANNELL, L. AND GIFFORD, R. 2010. Defining place attachment: A tripartite organizing framework. *Journal of Environmental Psychology 30,* 1, 1–10.

SCELLATO, S., MUSOLESI, M., MASCOLO, C., LATORA, V., AND CAMPBELL, A. T. 2011. NextPlace: A spatio-temporal prediction framework for pervasive systems. In *Proceedings of the 9th International Conference on Pervasive Computing*. Springer, 152–169.

SCHUSTER, D., ROSI, A., MAMEI, M., SPRINGER, T., ENDLER, M., AND ZAMBONELLI, F. 2013. Pervasive social context: Taxonomy and survey. *ACM Transactions on Intelligent Systems and Technology 4,* 3, 1–22.

SETTLES, B. 2012. *Active Learning*. Morgan & Claypool.

SMITH, I., CONSOLVO, S., LAMARCA, A., HIGHTOWER, J., SCOTT, J., SOHN, T., IACHELLO, G., AND ABOWD, G. D. 2005. Social disclosure of place: From location technology to communication practice. In *Proceedings of the 3rd International Conference on Pervasive Computing*. Springer-Verlag, 134–151.

STEVENSON, G., YE, J., DOBSON, S., AND NIXON, P. 2010. LOC8: A location model and extensible framework for programming with location. *IEEE Pervasive Computing 9,* 1, 28–37.

TAN, P.-N., STEINBACH, M., AND KUMAR, V. 2006. *Introduction to Data Mining*. Pearson, Boston.

TIWARI, M., MOHAN, P., OSHEROFF, A., ALKAFF, H., SHI, E., LOVE, E., SONG, D., AND ASANOVIĆ, K. 2012. Context-centric security. In *Proceedings of the 7th USENIX Conference on Hot Topics in Security*. USENIX Association, Berkeley, CA, 9–9.

VLACHOS, A. 2004. Active learning with Support Vector Machines. M.S. thesis, University of Edinburgh.

VU, L., DO, Q., AND NAHRSTEDT, K. 2011. Jyotish: Constructive approach for context predictions of people movement from joint Wifi/Bluetooth trace. *Pervasive and Mobile Computing 7,* 6, 690–704.

WANG, X., ROSENBLUM, D., AND WANG, Y. 2012. Context-aware mobile music recommendation for daily activities. In *Proceedings of the 20th ACM International Conference on Multimedia*. ACM, New York, 99–108.

WEISER, M. 1999. The computer for the 21st century. *Mobile Computing and Communications Review 3,* 3, 3–11.

YE, J., COYLE, L., DOBSON, S., AND NIXON, P. 2007. A unified semantics space model. In *Proceedings of the 3rd International Conference on Location- and Context-Awareness*. Springer-Verlag, Berlin, 103–120.

ZHENG, Y., ZHANG, L., MA, Z., XIE, X., AND MA, W.-Y. 2011. Recommending friends and locations based on individual location history. *ACM Transactions on the Web 5,* 1, 1–29.

ZHOU, C., FRANKOWSKI, D., LUDFORD, P. J., SHEKHAR, S., AND TERVEEN, L. G. 2007. Discovering personally meaningful places: An interactive clustering approach. *ACM Transactions on Information Systems 25,* 3, 1–31.

ZHU, X., GOLDBERG, A. B., BRACHMAN, R., AND DIETTERICH, T. 2009. *Introduction to Semi-Supervised Learning*. Morgan & Claypool.

ZHUANG, Z., KIM, K.-H., AND SINGH, J. P. 2010. Improving energy efficiency of location sensing on smartphones. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*. ACM, New York, 315–330.

A Place Recognition via Active and Semi-Supervised Learning

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.

A.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 6).

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 Google [2013] 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 $PL(t)$?

A.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 Tan et al. [2006]. 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: $G$, $W$, $B$, $GP$, $PL$, $t$
 1: $I \leftarrow \emptyset$                                        ▷ Training instances
 2: for all $p \in P$ do
 3:     $I_i^{pl} \leftarrow pl$                                     ▷ Class label
 4:     $t_i \leftarrow PL'(pl)$
 5:     ADDFEATURES($I_i, G(t_i), W(t_i), B(t_i), GP(t_i)$)
 6: end for
 7: $P_{model} \leftarrow$ TRAIN($I$)                               ▷ Any generic classifier
 8: ADDFEATURES($I_{test}, G(t), W(t), B(t), GP(t)$)
 9: $pl_t \leftarrow$ PREDICT($P_{model}, I_{test}$)
10: return $pl_t$                                                  ▷ Place at time $t$

 1: function ADDFEATURES($i, g, w, b, gp$)
 2:     $i^g \leftarrow g$                                           ▷ GPS features
 3:     $i^w \leftarrow w$                                           ▷ WiFi features
 4:     $i^b \leftarrow b$                                           ▷ Bluetooth features
 5:     $i^{gp} \leftarrow gp$                                       ▷ Google Place features
 6: end function
```

A.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

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

   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.

$$sim(I_1^w, I_2^w) = \frac{I_1^w \odot I_2^w}{\|I_1^w\| \times \|I_1^w\|}, \tag{2}$$

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 Lin [1998] to measure the similarity as follows.
$$sim(I_1^{gp}, I_2^{gp}) = \frac{2 \times IC(I_1^{gp} \cap I_2^{gp})}{IC(I_1^{gp}) + IC(I_2^{gp})}, \tag{3}$$
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.
$$sim(I_1, I_2) = \max_{f \in \{g,w,b,gp\}} sim(I_1^f, I_2^f). \tag{4}$$

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.

A.4 Active Learning

Platys Reasoner employs an active learning technique called *uncertainty sampling* Settles [2012]. 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.

A.5 Semi-supervised Learning

Platys Reasoner employs a semi-supervised technique called *self training* Zhu et al. [2009] 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,

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
 1: $L, U \leftarrow \emptyset$                                  ▷ Labeled and unlabeled instances
 2: BUILDDATASET($L, U, G, W, B, GP, PL, T$)
 3: if $L \neq \emptyset$ then
 4:     for all $u \in U$ do
 5:         $u^{sim} \leftarrow \max_{l \in L} sim(l, u)$
 6:     end for
 7:     $u_{uncertain} \leftarrow \min_{u \in U} u^{sim}$
 8: else
 9:     $u_{uncertain} \leftarrow$ Remove first instance from $U$
10: end if
11: return $u_{uncertain}$                                       ▷ An instance to label

 1: function BUILDDATASET($L, U, G, W, B, GP, PL, T$)
 2:     $i, j \leftarrow 0$
 3:     for all $t \in T$ do
 4:         if $PL(t) \neq \emptyset$ then
 5:             $L_i^{pl} \leftarrow PL(t)$
 6:             addFeatures($L_{i++}, G(t), W(t), B(t), GP(t)$)
 7:         else
 8:             addFeatures($U_{j++}, G(t), W(t), B(t), GP(t)$)
 9:         end if
10:     end for
11: end function
```

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:     $u^{pl} \leftarrow l_{nearest}^{pl}$                     ▷ 1-nearest neighbor
 7:     $L_{i++} \leftarrow u$
 8: end while
 9: return $L$                                                   ▷ All labeled 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
```

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.

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.

B 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 1) defined in Android Interface Definition Language (AIDL) declaring the functions a Platys-aware application can invoke Murukannaiah [2012]. 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 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 3) that returns a concrete implementation of the stub when an application binds with the middleware.

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

```java
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 4).

Listing 4: An application can bind to the middleware's service and receive a concrete implementation of the stub.

```java
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.