MobiSoC: A Middleware for Mobile Social Computing Applications

ANKUR GUPTA, ACHIR KALRA, DANIEL BOSTON, and CRISTIAN BORCEA
Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102, USA

Abstract. Recently, we started to experience a shift from physical communities to virtual communities, which leads to missed social opportunities in our daily routine. For instance, we are not aware of neighbors with common interests or nearby events. Mobile social computing applications (MSCAs) promise to improve social connectivity in physical communities by leveraging information about people, social relationships, and places. This article presents MobiSoC, a middleware that enables MSCA development and provides a common platform for capturing, managing, and sharing the social state of physical communities. Additionally, it incorporates algorithms that discover previously unknown emergent geo-social patterns to augment this state. To demonstrate MobiSoC's feasibility, we implemented and tested on smart phones two MSCAs for location-based mobile social matching and place-based ad hoc social collaboration. Experimental results showed that MobiSoC can provide good response time for 1000 users. We also demonstrated that an adaptive localization scheme and carefully chosen cryptographic methods can significantly reduce the resource consumption associated with the location engine and security on smart phones. A user study of the mobile social matching application proved that geo-social patterns can double the quality of social matches and that people are willing to share their location with MobiSoC in order to benefit from MSCAs.

Keywords: *Mobile social computing, middleware, smart phones*

1. Introduction

Social computing applications such as Facebook, MySpace, and LinkedIn improve social connectivity via collaboration and coordination by enabling compelling and effective on-line social interactions. However, these applications lead to a shift from physical communities to virtual communities. Currently, people living or working in the same places routinely miss opportunities to leverage inter-personal affinities (e.g., shared interests and backgrounds) for friendship, learning, or business through a simple lack of awareness. Furthermore, they are not aware of nearby places and social events, which they would normally like to visit or attend.

Mobile social computing applications (MSCAs) can take advantage of mobile computing algorithms, wireless technologies, and real-time location systems to help people re-connect with their physical communities and surroundings. With the widespread adoption of powerful mobile devices, such as smart phones, these applications will fundamentally change the way we interact with each other and with the physical world. Such applications can help people stay in touch anytime, anywhere, provide real-time recommendations about people, places, and events, or deliver customized/personalized content function of the user's geo-social context. For instance, MSCAs can answer questions such as the following: Are any of my friends in the cafeteria right now? Is there anybody who would like to play tennis nearby? Do people who work on wireless come often to this place? Which are the places where CS students hang out on campus?

MSCAs can be categorized into people-centric and place-centric. For example, a people-centric application can leverage geo-social patterns to provide enhanced social matching recommendations. The application uses real-time and historical location information, the social network graph, and basic user profiles to compute affinities between potential matches and deliver alerts to mobile devices. A place-centric example can be an ad hoc collaboration application that submits place-based queries (e.g., what's on the menu at the cafeteria?) to mobile users located in the desired place. In order to submit queries to the "right" people, this application identifies members of the requester's social network who are in the desired place, verifies their privacy constraints, and eventually forwards the query to some of them.

With research showing that users are increasingly willing to share their profile information and location in return for services [1], the time is ripe to start developing MSCAs. What is still missing, however, is a software platform to provide development and deployment support for coping with large user communities. Social computing applications in the Internet, such as MySpace and Facebook, have been very successful in the past few years because they attracted millions of users who generated a large amount of social content (e.g., profiles, photos, videos). Similarly, the very existence of MSCAs will depend on achieving a critical mass of users, who share their profiles, places, and real-time location information. To be ready to satisfy the demands of the users, if they are to use MSCAs, a software platform for mobile social computing will have to address a number of challenges.

First, it must provide mechanisms to: (*i*) capture the dynamic ties between users and between users and places; (*ii*) model, validate, and store these ties; and (*iii*) effectively share community data among multiple applications. Second, it is essential to provide infrastructure support to collect real-time user location in a scalable manner. Additionally, the location system must be chosen while considering trade-offs among accuracy, scalability, cost effectiveness, simple deployment, indoors/outdoors operation, and user control of location sharing. As location privacy is highly sensitive, this platform and the individual applications have to ensure that users cannot track each other.

Third, such a software platform should be capable of modeling the global state of a community and identifying emergent geo-social patterns. For example, users' mobility traces (i.e., location indexed by time) over a long period can be processed to learn their significant individual or group places. Furthermore, these places can be semantically enhanced by analyzing the user-generated tags associated with them. Finally, this platform must help mobile devices reduce their energy consumption. Trade-offs between local execution on mobile devices and offloading application components to servers, which results in extra communication costs, must be considered. For instance, the software platform should run the common computationally intensive tasks required by multiple applications or multiple users. Additionally, data caching and aggregation should be provided in order to reduce both the computation and communication costs.

This article presents MobiSoC, a mobile social computing middleware that provides support for programming and deploying MSCAs, while addressing the challenges mentioned above. MobiSoC offers a common platform for capturing, managing, and sharing the social state of physical communities. This state is composed of people profiles, place profiles, people-to-people

affinities, and people-to-places affinities [2]. The social state evolves continuously over time as new user profiles, social ties, place-related information, or events are created. Additionally, the consistent view of the social state provided by MobiSoC enables algorithms that discover previously unknown emergent geo-social patterns (i.e., people-to-people and people-to-places affinities), which can further augment the state. MobiSoC runs on trusted servers and provides a simple API for developing MSCAs. To improve the responsiveness and energy efficiency on mobile devices, each MSCA is split into an MSCA service that runs on top of MobiSoC on regular servers and a thin mobile client that interacts with the service and MobiSoC over the Internet.

MobiSoC was implemented using an extensible service-oriented architecture. Its core modules were implemented as Java services that run over the Apache Tomcat server. These services are exposed using KSOAP, a SOAP toolkit designed to work with Java J2ME on mobile devices. We chose Intel's PlaceLab location engine, which computes location on mobile devices using the position and signal strength of visible WiFi access points, because it works both indoors and outdoors and allows users to control location sharing. We validated MobiSoC by building two MSCAs on top of it: Clarissa, a location-based mobile social matching application, and Tranzact, a place-based ad hoc social collaboration application. These applications were tested successfully on Windows-based smart phones, which connect to the Internet over WiFi.

The experimental evaluation quantified both the server-side and phone-side performance. The server-side results demonstrated that user-perceived response times in the 500 ms range can be achieved for a population of 1000 users. We also showed that these results can be significantly improved through data caching. Furthermore, the GPI algorithm [36] incorporated in MobiSoC successfully identified geo-social patterns. Specifically, it used one month of mobility traces collected from smart phones carried by students and faculty on our campus to identify all ad hoc social groups that met regularly during that period.

The experiments on the phone showed that the location engine and the optional security primitives are the most expensive considerations in terms of processing and battery power. To alleviate resource consumption, we designed and implemented an adaptive method for computing and updating the location on the phone, which improves the running time by as much as 8 times. Our experiments also showed that RSA (out of 5 cryptographic methods tested) achieves the best performance in providing integrity and authentication for location updates. AES with or without RSA encryption of its key cipher proved to be the best method to provide confidentiality for larger messages.

The user study of Clarissa with 13 mobile users and over 40 on-line users demonstrated that location and geo-social patterns can double the quality of social matches. Finally, they also showed that people are willing to share their location with MobiSoC, although not at room-level granularity, in order to benefit from MSCAs.

The rest of this article is organized as follows. To provide the necessary context for our work, we start with a discussion of background and related work in Section 2. Section 3 describes the MobiSoC design and explains its core modules. Section 4 explains how to build applications over MobiSoC and illustrates that with two prototype applications. Section 5 presents the prototype implementation, and Section 6 shows its experimental results. The article concludes in Section 7.

2. Related Work

Recently, we have witnessed an increasing number of social applications on mobile devices that go beyond mobile versions of Facebook or MySpace. One category of such applications signals matching interests between people using spontaneous one-hop ad hoc communication. This category includes Nokia sensor [3], Social Net [4], and Social Serendipity [5]. Another class of applications leverages the absolute location of the user to provide location-based services. Examples include downloading songs currently playing at a place [6], posting and reading place-linked virtual notes [7], delivering nearby matches based on SMS-uploaded location [8], or showing nearby businesses or ATMs [9]. However, all these applications represent just the tip of the iceberg. Many novel applications will be enabled by geo-social information and emergent community patterns.

While such applications can be developed from scratch, having programming support from a middleware or framework can improve significantly the productivity of developers as well as the performance and features of the applications. MobiSoC is among the very few platforms that target mobile social computing. The closest work to ours is the SAMOA middleware [10], which extracts, and shares with applications, location-aware social networks using semantic-based context-modeling and matching algorithms. Both middleware architectures use places and user profiles to model the social world. The main difference is that MobiSoC takes a community-centric approach, while SAMOA takes a user-centric approach (i.e., it extracts social networks for individual users). These choices have implications in the system design, trust, and the features provided to applications. SAMOA uses a mix of distributed and centralized techniques, and MobiSoC is purely centralized. In this way, we can maintain a complete social state of a community and identify emergent community patterns. Similar functionality would be much harder to provide with SAMOA, where the social state is partitioned in a user-centric fashion. Furthermore, a trusted centralized entity helps when providing privacy guarantees. Finally, since MobiSoC focuses on communities, it can support a larger spectrum of MSCAs through a richer API. Two other projects that share with us the goal of providing elements of social context to mobile users are Active Campus [11] and the Whereabouts Diary [12].

Unlike these projects, MobiSoC identifies emergent community patterns by analyzing geo-social relationships between people and places. In this aspect, we can leverage ideas from recent social network analysis studies [13]. For example, patterns such as power-law/small-world topology have been found in networks ranging from high school friendships to citation networks in sciences [14]. Applying this type of social knowledge to system design has been recently considered in peer-to-peer and mobile systems [15, 16].

People-centric sensing is a direction similar to our research that has recently emerged. The idea is to use smart phones or vehicular systems as mobile sensors. MetroSense [17] proposes a three-tier architecture for scalable support of concurrent people-centric sensing applications. Participatory Sensing [18] seeks to build short-term, community-oriented urban sensor networks. Micro-Blog [19] is a participatory sensing application that allows people to use their smart phones to generate and share geo-tagged multimedia. MyExperience [20] is a system that captures and shares both user-specific and device-specific data on smart phones.

MobiSoC is designed to recognize the fact that sharing user context data with applications raises significant privacy concerns, with location data being especially sensitive and vulnerable to privacy attacks. In [21], the authors present an access control algorithm based on access-rights graphs to avoid such indirect privacy violations caused by context-sensitive applications. Confab [22] is a generic toolkit that facilitates the development of privacy sensitive ubiquitous applications. SmokeScreen uses opaque identifiers, which can be resolved only by a trusted broker, for presence sharing among strangers [23]. AnonySense is an architecture for people-centric sensing that protects user privacy through anonymity [24].

While social context, and especially relations between communities and their places, have rarely been incorporated in middleware platforms, frameworks have been developed to support the acquisition, representation, delivery, and reaction to context information [25]. Middleware for developing applications in ubiquitous computing environments include GAIA [26] and One.world [27]. Finally, context-aware middleware such as Migratory Services [28] and MUM [29] have been developed for mobile ad hoc networks as well.

3. MobiSoC Architecture

This section presents the design of MobiSoC, our mobile social computing middleware. A key goal of MobiSoC is to capture and manage the social state of a community and to learn emergent social state information as illustrated in Figure 1. The basic social state of a physical community is composed of people profiles, place profiles, social ties between people, and associations between people and places. This state evolves continuously over time as new user profiles, social ties, place-related information, and events are created. Additionally, learning algorithms can determine people-to-people and people-to-places affinities, and based on these affinities discover previously unknown emergent geo-social patterns. The newly discovered information can be used to augment the user profiles and the characteristics of the places.

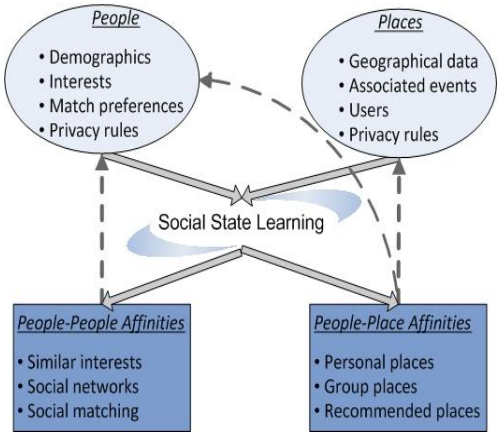


Figure 1. Social State of a Community

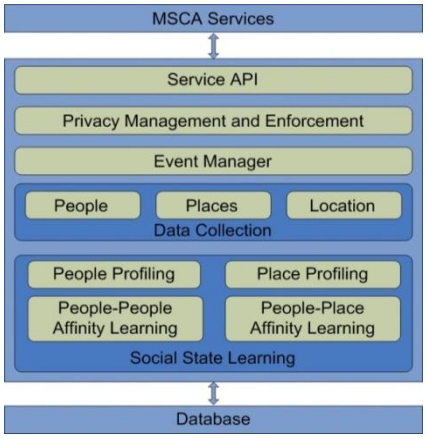


Figure 2. MobiSoC Architecture

MobiSoC acts as a centralized entity for social state management and provides a service API to programmers for application development. We chose a centralized solution because it is simpler to maintain a consistent view of the social state and to provide access control to privacy sensitive data. In a potentially distributed (or even ad hoc solution), the social state could be partitioned

among many systems, making it difficult to identify community patterns. MobiSoC's consistent social state enables community-centric algorithms for extraction of geo-social emergent patterns. From a trust point of view, a centralized architecture allows for uniform privacy enforcement by the trusted entity. Additionally, having servers in the system architecture helps to improve the response time and battery lifetime on the mobile devices as certain parts of the applications can be executed at the server side. The MobiSoC architecture is presented in Figure 2. The internal modules can be physically distributed on multiple servers in order to achieve scalable operation. In the following, we present each of the middleware modules.

3.1 Data Collection

The *People* sub-module allows applications to collect, store, and modify user profiles. Besides basic demographic information, users can provide information regarding social interests, preferences, and social ratings. This sub-module has also mechanisms to introduce new groups and add new social contacts, and it maintains a social network based on this information. The *Places* sub-module supports the collection of geographical data and maps for buildings, offices, and outdoor locations. Furthermore, it provides mechanisms to introduce and modify social events associated with a place. The *Location* sub-module receives and stores location updates from the mobile devices. We decided to determine the location of the mobile device on the device itself for privacy reasons. We believe that users would be very concerned if a certain hardware/software infrastructure would track them to determine their real-time location. Therefore, we allow them to control when and how often location updates are sent to our middleware.

3.2 Social State Learning

This module learns emergent social state information. Using an analogy from relational databases, the social state module discovers and represents relationships between two entities, person and place. The *People Profiling* sub-module is used to provide user-centric information to services such as profiles, social links, and social groups. Additionally, this module enhances user profiles based on newly discovered information about individual users. For example, this module could find out that a user attends research seminars regularly, plays tennis every Friday, or works together with another user. Similarly, the *Place Profiling* sub-module shares place-centric information and enhances the semantics of the place with social information. For instance, this module could find out how crowded a place is at different times in the day, popular social events which happen at that place, or the demographics data of people who visit the place frequently.

The *People-People Affinity Learning* sub-module computes social affinities based on factors such as similar interests, similar backgrounds, common friends, or common places. These affinities are computed for each pair of users. For example, similar interests can be discovered if people visit the same place at different times. As the user and place profiles change over time, these affinities are re-computed periodically. The *People-Place Affinity Learning* sub-module analyzes users' mobility traces to identify significant places for individuals or groups. To discover these geo-social ties between people and places, it performs temporal synchronization on the mobility traces and uses clustering techniques to determine repeated user co-presence at a place.

For instance, informal ad hoc social groups can be discovered based on co-location in the same place at the same time. Examples of such groups are members of the faculty that routinely have lunch together or students that meet in a game room every Friday.

3.3 Event Manager

This module is used for asynchronous communication with the applications. The applications or the middleware services can register events with the middleware to receive notifications when a certain part of the social state changes. For example, an application might want to be notified in real-time of the co-location of two users, a user presence at a given place, or a new social match based on newly identified social affinities. Social state changes are detected via event triggers that include time, location, and co-location, as well as external triggers such as the generation of a new social match.

We decided to let the mobile devices pull their events from the middleware instead of having the middleware push the events onto them. There are two reasons for this decision. First, mobile devices frequently change their IP addresses, and the middleware would have to be aware of them. Second, the devices might not have Internet connectivity all the time because power saving mechanisms might turn off certain wireless interfaces or the users might turn off the devices themselves. In such situations, the middleware would have to keep track of the status of mobile devices, on-line or off-line. In our architecture, the mobile devices periodically poll the middleware for event notifications. To reduce useless communication which could impact the battery lifetime, our implementation, as illustrated in Section 4, takes advantage of the location engine running on mobile devices to retrieve event notifications during the location updates.

3.4 Privacy Management and Enforcement

This module manages and enforces privacy rules on behalf of the entities in the system (users and applications). The privacy rules provided by these entities are stored in the database. We had the option to enforce these rules at the database level by using restrictive queries [30], but complex and evolving geo-social privacy constraints are difficult to enforce with such queries. For instance, a user might want to have different location sharing rules as a function of the strength of social ties, group membership, her current location, or time of the day. Additionally, a user might want to be prompted if another user requests access to her information.

In MobiSoC, applications register privacy preferences with the middleware on behalf of their users. These preferences are expressed in the form of a privacy statement, which has a primary entity that issues the statement and a secondary entity on which the statement applies. Unlike the primary entity which is always an individual user, the secondary entity can be individual users, groups of users, or applications. An MSCA that requests information pertaining to a primary entity in order to share it with a secondary entity must call the middleware to verify the associated statement (note that these statements are shared among all MSCAs). Subsequently, it grants/denies access or forwards the request to the primary entity that can further allow or disallow information access. Section 5 presents the details of this module.

4. Application Development over MobiSoC

This section presents the general structure of any MSCA built on top of MobiSoC, the MobiSoC's development API, and code examples for two prototype applications, Tranzact and Clarissa, successfully developed and tested on smart phones.

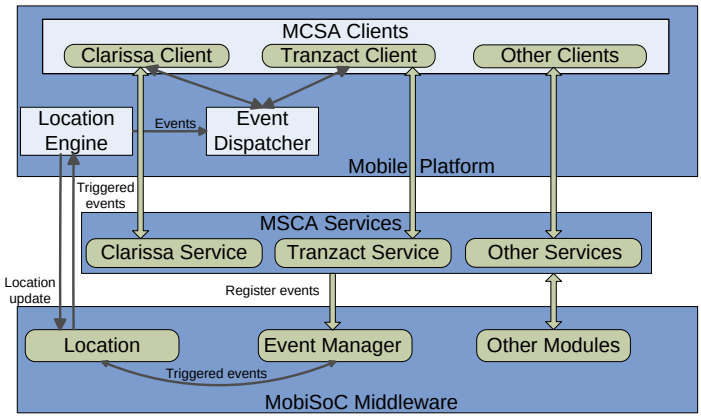


Figure 3. Building Applications over MobiSoC

4.1 Application Structure

Each MSCA is split in two parts: (1) an MSCA service that runs on servers and accesses social state information using the MobiSoC's service API, and (2) a thin MSCA client that interacts with the MSCA service over the Internet. Figure 3 shows how our prototype applications are divided into client and service parts. This application structure improves the responsiveness and energy efficiency on mobile devices. Essentially, the MSCA services offload the computationally intensive components of the applications on the servers. In this structure, MSCA clients cannot interact directly with the middleware; they can only interact with their associated services. Therefore, the programmers are naturally forced to design the applications in the required structure.

Another benefit of this structure is that services can easily maintain global state across the mobile clients. For instance, services can compute and cache results that are requested by many clients. Similarly, they can maintain service-specific real-time information about an entire community. In this way, the middleware overhead is reduced significantly (and the response time improved) as individual requests can be answered without invoking the middleware for each of them. Finally, services running on trusted servers are implicitly more trusted that applications running on mobile devices. For instance, it is preferable to have services, instead of individual clients, retrieve information about individual users and compute aggregates.

MSCA clients can communicate synchronously (i.e., request/reply) with the services. However, many times they need to be contacted when certain social, temporal, or geographical conditions are met. To achieve this goal, the MSCA services register events with the middleware, which will deliver them to clients. As illustrated in Figure 3, the event notification delivery is done periodically when the mobile clients update their location with the middleware. Although this mechanism introduces a one location update period delay, it improves the energy efficiency on the

mobile devices. Once the location engine receives events from the middleware, it passes them to an Event Dispatcher that subsequently delivers each event to its target MSCA client.

Module	Function	Module	Function
Event Manager		**Social State Learning**	
	registerEvent	Place Profiling	searchPlaces
	deleteEvent		getPlaceInfo
Data Collection			getPlaceSocialGroups
People	createAccount		getPlaceAttendancePatterns
	updateProfile		getPlaceDemographicsPatterns
	requestSocialContact		getNearbyPlaces
	addSocialContact		searchSocialEvents
	createSocialGroup		getSocialEventInfo
	addSocialGroupMember		getSocialEventsPlaceHistory
	requestSocialGroupMembership		getNearbyEvents
Places	setPlaceData	People-Place Affinity	getUserPlaceHistory
	setPlaceTag		getGroupPlaceHistory
	addSocialEvent		getPeopleAtPlace
	addSocialEventMember		getSocialGroupsAtPlace
	requestSocialEventAttendance	People-People Affinity	getCommonInterests
Location	setUserLocation		getCommonSocialGroups
Social State Learning			getCommonSocialContacts
People Profiling	searchProfiles		getSocialNetworkDistance
	getProfileInfo		getCoPresenceHistory
	getSocialContacts		getAffinityMatrix
	searchSocialGroups	**Privacy Manager**	
	getSocialGroupInfo		setPrivacyStatement
	getUserSocialGroups		deletePrivacyStatement
	getUserLocation		checkPrivacyConstraints

Table 1. Middleware API

4.2 API and Code Examples

The current API exposed to MSCA services is presented in Table 1. Since most of the function names are self-explanatory, we provide a very brief overview of the main categories of functions. The event manager API allows services to register and delete events. The data collection API is used to store data about people (profiles, social ties, and social groups), places (physical descriptions, user generated tags that characterize them, and associated events), and location. The people profiling API provides access to people-centric data such as searching profiles by tags and keywords or retrieving the social groups associated with a given user. Furthermore, the data returned by this API could contain profile data mined by the middleware (e.g., emergent patterns). Similarly, the place profiling API offers access to place-centric data such as attendance and demographic patterns or history of social events that happened at a given place.

The people-place affinity API provides information about the emergent visiting patterns at a given place by individual users or groups, as well as real-time data about the occupants of a place. The people-people affinity API can be invoked to retrieve social connections between two users. Specifically, it can return an affinity matrix between two users, computed across several geo-social factors, common social groups and ties, or the co-presence history. The privacy manager API allows services to set and delete privacy statements as well as to check privacy constraints.

As more applications will be developed over MobiSoC, we expect to add new functions or even to update existing ones. Besides the service API, we also provide a very limited API for MSCA clients on mobile devices to check the current location, enable and disable the transmission of location data to the middleware, and listen for events from the local event dispatcher.

```
contactMatches ← getSocialContacts(requester)
potentialTargets ← getPeopleAtPlace("Cafeteria")

For each user in contactMatches ∩ potentialTargets
    pAction ← checkPrivacyConstraints(user, requester, "TzEvents")
    If pAction == Allow
        tzEvent.setTimeConstraints(now)
        tzEvent.setLocationConstraints("Cafeteria")
        tzEvent.setTargetUser(user)
        tzEvent.setDescription("Tranzact" + request + requester)
        registerEvent(tzEvent)
```

Figure 4. Tranzact Pseudo-code

Tranzact is an application for place-based ad hoc social collaboration. Its clients send queries for real-time information from various places. Figure 4 shows the processing done by the Tranzact service when it receives such a query. For instance, the requester might want to find out the current menu at the cafeteria (which is not posted anywhere outside the cafeteria). In order to answer the query, while not bothering strangers, Tranzact starts by identifying the social contacts of the requester who are currently in the cafeteria. This task is achieved through two function calls to the People Profiling module and People-Place Affinity module. Before sending the query, Tranzact invokes MobiSoC to verify if the potential destinations are willing to accept events from this application on behalf of the requester. The available users receive the request using our event-based communication mechanism. Responses are sent back through the same mechanism.

```
contactMatches ← getSocialContacts(requester)
userGroups ← getUserSocialGroups (requester)

For each group in userGroups
    groupInfo ← getSocialGroupInfo(group)
    groupMembers ←groupInfo.getMembers()
    contactMatches ← contactMatches ∪ groupMembers

For each user in (allUsers – contactMatches)
    affinityMatrix ← getAffinityMatrix(requester, user)
    higherWeights ← {"Sports", "Music"}
    matchScore ← computeScore(affinityMatrix, higherWeights)
    If matchscore > threshold
        affinityMatches.add(user)

For each user in affinityMatches ∪ contactMatches
    matchEvent.setTimeConstraints(2pm,4pm)
    matchEvent.setCoPresenceConstraints(requester, user)
    matchEvent.setTargetUser(requester)
    matchEvent.setDescription("Hangout  Match" + user)
    registerEvent(matchEvent)
```

Figure 5. Clarissa Pseudo-code

Figure 6. Clarissa matching UI

Clarissa is a location-based mobile social matching application. Figure 5 illustrates the processing done by the Clarissa service when a student has a two hour break between two classes and is looking for a hangout partner on campus. This person must be available between 2PM and 4PM, and she must be in close proximity of the requester. Additionally, she has to be either someone known by the requester or someone who shares common interests (in this example, we look for sports they play and music preferences). The service gets the union of known people, social contacts and members of common groups from the People Profiling module. It then computes a matching score with all the remaining users. This score is computed by assigning higher weights to certain affinity factors (i.e., sports and music). The raw affinity scores are

retrieved from the People-People Affinity module. Once the potential matches are identified, Clarissa registers events for the requesters, which are triggered by the co-presence with potential matches during the specified time interval.

5. Prototype Implementation

We built a prototype implementation for MobiSoC using a service-oriented architecture, which supports evolution by providing modularity, extensibility, and language independence. This prototype was used to develop several applications, including Clarissa and Tranzact. Figure 6 shows Clarissa's matching user interface on a smart phone. The core middleware modules are implemented as services and written in JAVA. They run over the Apache Tomcat application server and store data in a PostgreSQL database, which provides good support for GIS data such as maps and place related information. MSCA clients run on *iMate* and *HTC* smart phones, which have the Windows Mobile operating system. WebSphere Everyplace Micro Edition Java (WEME) [31] was selected as the Java Micro Edition implementation, with Personal Profile 1.0 and MIDP 2.0 as the minimum required configuration.

We had a number of options for communication between client applications on mobile nodes and the application services on the server. Since our services are written in JAVA, the easiest way would have been Java serialization, but this option does not provide for language independence. The next option considered was to use a lightweight parser [32] to convert JAVA objects to XML and then transport these objects over TCP. Unfortunately, we found compatibility issues between the JAVA mobile edition library (J2ME) and the parser. Finally, we decided to expose our services via the Simple Object Access Protocol (SOAP). SOAP offers language independence, and SOAP clients are available for many popular languages. Additionally, it provides a clean transport mechanism, with the client and services communicating over HTTP. We use the KSOAP J2ME [33] library that implements a subset of SOAP 1.1 and has a memory footprint less than 50KB. This makes it extremely suitable for resource constrained devices such as smart phones. Furthermore, the KXML parser offers good performance comparable to XML-RPC, yet provides support for custom data types.

5.1. Location Engine

The location engine on the clients is a modified version of Intel's Place Lab software [34]. This engine estimates the user location based on the location and signal strength of the WiFi access points visible from the mobile device. Each mobile device holds a database consisting of access point names and their locations in the area of interest. In the basic centroid method, when a mobile device receives beacon messages from the visible access points, it retrieves each access point's coordinate from the database and computes the estimated user position by averaging the location of the access points.

We also implemented a fingerprinting based algorithm, similar to the one described in [35], to improve the location accuracy. This algorithm creates a database of known location points; for each location point, the signal strength of all visible access points is recorded as well. At runtime, the algorithm measures the signal strength of the visible access points at the current location, and

then identifies the closest known points using a distance function based on signal strength. The estimated current location is then computed as the average location of these points. Although this algorithm has generally improved accuracy (as demonstrated in Section 6), we preferred to use the centroid method for two main reasons. First, creating a database of known points does not scale for larger regions because the algorithm requires a high density of such points (e.g., one every 6 square meters) and their recorded location must be highly accurate. Second, fingerprinting trades off battery power for accuracy as it employs significantly more computation. We believe battery power is more important than high location accuracy for most applications developed over MobiSoC.

5.2. People-People Affinity

To computes social affinities between people, our middleware needs user profiles and mobility traces. To speed up experimentation, we decided to leverage the Facebook profiles that most students on our campus already have. This module computes affinities between every pair of users along several social factors such as common events, mutual friends, common interests, common places, as well as geo-social factors such as presence in a place or co-presence. Data such as common events or mutual friends is accurately captured from the user profiles. In our current implementation, however, the background and interest scores are based on word matching and could result in lower values sometimes; this is because the profiles do not employ a well-defined, fixed vocabulary. Note that these affinity scores are generic in nature (computed on a scale from 1 to 1000 along each factor), and individual MSCAs may decide to assign higher weights to the criteria that are more important for the application. For instance, if a user is looking for a tennis partner with similar background, then the "sports" factor is assigned the highest weight, followed by the background data.

Using geo-temporal data to measure user affinities is non-trivial, but certainly desirable. Prior research proves the validity of using mobility traces to calculate user affinities [4, 7]. However, little has been done to quantify the results and effectively use them in any meaningful manner. We devised algorithms that study patterns in users' geo-temporal data to compute and quantify social affinities. For example, the co-presence component measures how much time two users spend in proximity of each other. Similarly, the criss-cross component measures how many times the paths of two users crossed each other even if the users have not spent substantial amount of time together (e.g., familiar strangers).

PostgresSQL/PostGIS simplify the implementation of these algorithms because they provide a simple mechanism for storing location data and translation between human readable labels and GIS coordinates. Furthermore, location is stored in a binary format that allows indexing and optimizations for quick searching. For example, getting a list of all users in a building does not require the middleware to check every user in the database, but it is optimized to search only within the place associated with that building.

5.3. People-Place Affinity

Mobility traces can be used not only to infer People-People affinities as in the previous section, but also different types of People-Place affinities. The first algorithm, which we developed for the People-Place module, is GPI, an algorithm that learns previously unknown ad hoc social groups and their associated meeting places. The middleware API can be used to retrieve GPI's results, subject to privacy constraints, for geo-social recommendation applications. For instance, new students could learn about popular hangouts on campus or faculty could learn about students attending research seminars on certain topics.

So far, mobility traces have only been used in algorithms that identify significant places for individual users [37, 38]. To the best of our knowledge, no work has been done on using community mobility traces to identify social groups and places that have importance for a group of people. While place identification algorithms typically deem a place significant based on repeated patterns of user's presence at the place, identifying group members and group-place associations is much harder because informal groups do not have a clear pattern in terms of group meeting times, group composition, or group member attendance.

Therefore, GPI relies on repeated user co-presence at the same place to determine the group members, and consequently the meeting places. The underlying assumption is that group members have a much higher degree of co-presence (DCP) than non-group members. First, GPI performs time based clustering on these traces to infer place-wise co-presence between users. Next, to determine if a given place is a group hangout, it checks the users' DCP with respect to the total number of visits to infer potential groups and their respective hangouts. However, the fact that group members are typically present only at a fraction of the meetings and non-group members can possibly be present at meetings raises the following question: What is the required DCP between group members considered by GPI? We performed a theoretical analysis that determined the optimal required DCP that allows GPI to balance the trade-off between group member identification percentage and false positives percentage. The theoretical analysis and simulation results demonstrated that 90%-96% of group members can be identified with negligible false positives when the user meeting attendance is at least 50% [36]. Experimental results using one month of mobility traces collected from smart phones carried by students and faculty on our campus successfully identified all groups that met regularly during that period. Additionally, the group places were identified with good accuracy. Results and discussions about GPI's scalability are presented in the next section.

5.4. Privacy Management

The privacy management module allows MSCA services to register privacy statements on behalf of the users (in the current prototype, the statements are stored in the database), and then verifies these statements when necessary. The structure of the privacy statement, shown in Figure 7, includes access control objects, information objects, and action objects. An access control object defines the conditions under which a statement applies. Currently, these conditions are user's location, co-location with other users, time of the day, or a combination of these constraints. The information object defines the information that is restricted, and we currently support restrictions

over location, events, profile data, and social network data. The action object describes the action to be taken in case the secondary entity tries to access the information defined by the information object. Our prototype supports denial of access and sending an appropriate message to the secondary entity, or forwarding the request to the primary entity that can further allow or disallow information access. More sophisticated actions, such as removing only the privacy sensitive data from an object, will be added in the future. Additionally, we plan to design mechanisms to control information leakage through inference.

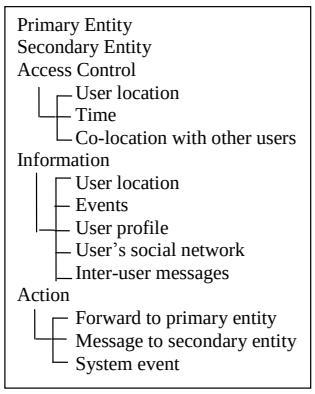


Figure 7. Privacy Statement Structure

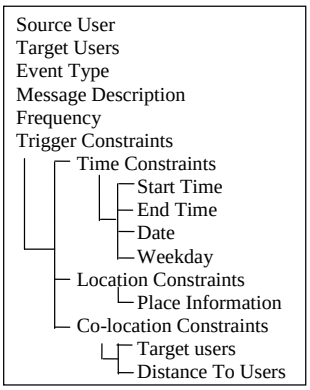


Figure 8. Event Structure

Each time an MSCA service needs to share information pertaining to a primary entity with a secondary entity, privacy statements are fetched from the database with respect to the primary entity, secondary entity, and the information requested by the secondary entity. Next we check the access control object for each of these statements to determine if the statement is currently applicable based on the primary entity's location, co-location with other users, and time of the day. If such a statement is found than we proceed with the action defined in the statement.

For example, the Tranzact service allows a student to specify that she is willing to receive requests only from her advisor when she is in the cafeteria between 1PM and 3PM. The Tranzact service generates a privacy statement wherein the primary entity is the student and the secondary entity is the advisor. The access control object sets "location=cafeteria" and "time=1PM-3PM". The information object specifies TzEvents under the "Events" category, and the action object sets "Access Allowed" under the System Event category (by default, the action for TzEvents is "Access Denied"). Let us now assume that this student is selected among the potential candidates for a Tranzact request while being in the cafeteria at 2PM. The Tranzact service requests a privacy check from MobiSoC, as was shown in Figure 4. The middleware fetches from the database the privacy statement where the primary entity is the student, the secondary entity is the requester, and the event is TzEvents. Then, MobiSoC checks the constraints and returns the associated action. For our example, it returns "Allow" if the requester is the advisor; otherwise, it returns "Deny".

5.5. Event Management

Figure 8 illustrates the structure of an event. Besides the spatial-temporal constraints, this structure includes the type of event, the source and target of the event, a message description that the user receives when the event is delivered, and the frequency of sending the event. Once an

event is registered with the event dispatcher, all the associated information is stored both in memory (for fast processing) and in the database (for fault-tolerance). For each registered event, the event manager keeps track of its time constraints by setting a timer that is fired when these constraints are satisfied. Optionally this could be done by maintaining a minimum priority queue that stores events based on their activation time and then running a single thread that wakes up when the next event in the queue is to be activated.

Once the time constraints are satisfied, the event manager checks if the event has location or co-location constraints. If so, the event is pushed onto a hashtable that stores location-based events; otherwise, the event is triggered. A thread periodically checks all the events in the hashtable for conformance with the location-based constraints and triggers the events once these constraints are satisfied. When the client devices communicate with the location update module, the event is collected from the event manager and delivered to the dispatcher at the client side.

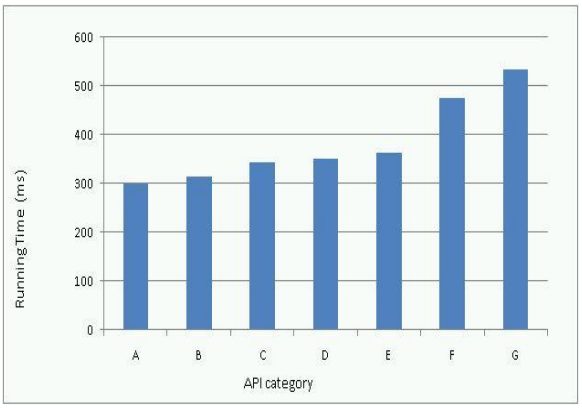
6. Evaluation

The first evaluation of our prototype was the successful implementation and deployment of Tranzact and Clarissa. These applications have been tested with 10+ mobile users carrying smart phones on our campus. Their development proved that complex MSCAs can be rapidly implemented on top of MobiSoC. Subsequently, we performed an experimental study that had three goals. The first was to understand the server-side performance of our middleware. Specifically, we wanted to see if the architecture can provide good real-time performance for a large population of users and to identify potential bottlenecks. The second goal was to investigate which factors impact the performance of applications on smart phones the most. Since the MSCA computation is mostly done by the MSCA services at the server-side, the main factors remain the location engine and the optional security primitives used to provide message confidentiality, integrity, and authentication. The third and final goal was to perform a user study for one of our applications, Clarissa, and understand its social usefulness. Furthermore, we investigated the benefits of geo-social patterns on social matching and the potential impact of privacy concerns on mobile social computing. The experiments used WiFi-enabled HTC Wizard and iMate KJam smart phones, with a 195MHz ARM processor and 100 MB RAM. The middleware and the associate services ran on a Pentium 4, 2.66 GHz machine with 960 MB RAM.

6.1 Server Side Experiments

MobiSoC's scalability and its impact on user perceived response time are essential for supporting MSCAs, which often have a large user base. The first category of server-side experiments evaluates the running time of the middleware API for a relatively large user population. Since our prototype does not have enough users yet to test scalability, we generated data for 1000 mobile user profiles, 100 places, 500 groups, and 500 events. Each user has approximately 150 random friends (i.e., 150 is the Dunbar's number [39] for the size of a true social network). The mobile device of each user updates its location every 10s from one of the 100 places. The user spends a randomly selected time in a place and then moves to another place. The location data is collected for eight hours per day for 30 days.

Figure 9 presents the running time for common middleware API functions provided by MobiSoC's Social State Learning module. These functions are executed in real-time and are the most computationally expensive in our middleware. The results, between 300ms and 500ms, are heavily impacted by accesses to the database to fetch community data. To understand how these values influence the user perceived response time, we profiled two MSCA services built on top of MobiSoC, namely Tranzact and VoteCenter. Since Tranzact was already presented, we continue with a brief description of the VoteCenter application. This application allows members of a group to vote on issues of common interest when they visit places associated with the group. Group members receive poll queries when they hit their respective places as long as their privacy settings allow it. Each poll has a maximum number of votes or a maximum time after which it expires. Any group member can start a poll and vote. For example, students in a lab could ask when the next research group meeting should be, and this question is only delivered to research group members when they enter the lab.



A	B	C	D	E, F, G
searchProfiles	getNearbyEvents	getCommonContacts	getProfileInfo	getUserPlaceHistory
searchEvents	getNearbyPlaces	getCommonGroups	getSocialEventInfo	getUserPlaceHistory
searchGroups			getSocialGroupInfo	getCopresenceHistory
searchPlaces			getPlaceInfo	

Figure 9. Running Time for Common API Categories

Figure 10 plots the average distribution of middleware API invocations per service request. The total numbers of 1.7 and 2.4 result in a user perceived response time of less than 1s. These relatively low numbers demonstrate the advantages of using MobiSoC's high level primitives that hide the complexity of geo-social state learning from programmers. We consider the response time of less than 1s to be acceptable for mobile users. In fact, the distribution shows that the social state learning API, which is the most expensive, has around one invocation per service call. The rest goes to the other modules, Data Collection, Event Manager, and Privacy Manager, respectively. Since the Data Collection API is rarely invoked (except for updating user's location), we focused next on the cost of event and privacy management. The average cost of a call to each of them is 50ms, which roughly corresponds to a typical database access. If the privacy statement needs to fetch location information, the average privacy verification cost is 100ms. Therefore, for the two profiled applications, we conclude that the expected response time for a user is in the 500ms range. To scale to a larger number of users, one can take advantage of the service-oriented nature of

MobiSoC and distribute its functionality among multiple servers (note that our experiments used a regular workstation for the application services, MobiSoC, and the database).

As observed by the designers of Internet-based social applications, fast access to the database is crucial for good response time. For instance, Facebook uses MemcacheD [40], a high-performance distributed memory object caching system that boosts their performance by 20%. We plan to use a similar solution to improve MobiSoC's performance. However, the caching will be done per MSCA service. Furthermore, unlike Internet-based social applications, MobiSoC has to deal with the issue of frequent location updates from large populations of users. The profiling of the two applications presented above revealed that updating the location in the database happens two orders of magnitude more frequently than API invocations. Therefore, our first global optimization is to cache the location updates and use an asynchronous propagation to the database. Figure 11 demonstrates an order of magnitude improvement for 625 concurrent updates. This caching mechanism also improves the performance of the Event Manager (i.e., many events are triggered by location or co-location) and the Privacy Manager (i.e., many constraints are location-based).

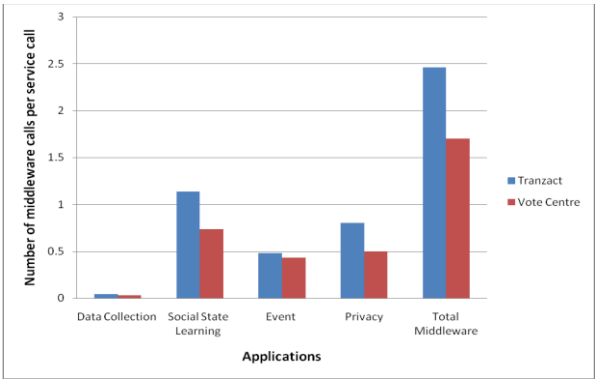


Figure 10. Service Profiling: Distribution of Middleware Calls for Tranzact and Vote Center

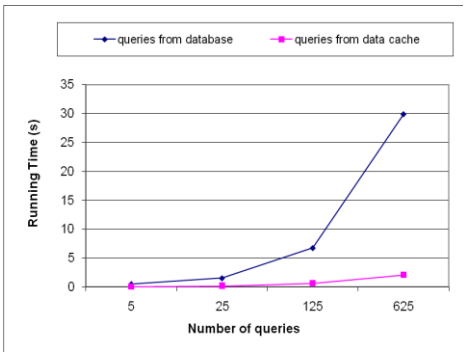


Figure 11. Location Querying: Database vs. Data Cache

The second category of server-side experiments evaluates the off-line social-state learning algorithms that analyze large amounts of historical data. Although their performance does not affect the user perceived response time, we believe that these experiments can provide valuable lessons for real-life, large scale deployments. The People-People affinity learning algorithm runs in two steps. The first step involves sorting location points by time and mapping the location points to places for each user. This step is performed by issuing database queries and takes most of the total running time. The second step is much faster as it involves analyzing the extracted location data to determine affinity scores along the location-based factors (e.g., co-presence, criss-cross) as well as computing scores along other factors (e.g., interests, background). Figure 12 shows the running time of the affinity learning algorithm between two users function of the number of location points (e.g., 35000 points could correspond to about two months of mobility traces, with a 1 minute update frequency). These results include both steps, but the second step takes only 20ms for 35000 location points. Nevertheless, they seem to indicate a high running time for computing the affinities between all user pairs. However, only the fast second step is executed

N^2 times, where N is the number of users. The first step is executed only N times. Therefore, our algorithm can finish in several hours for hundreds to thousands users.

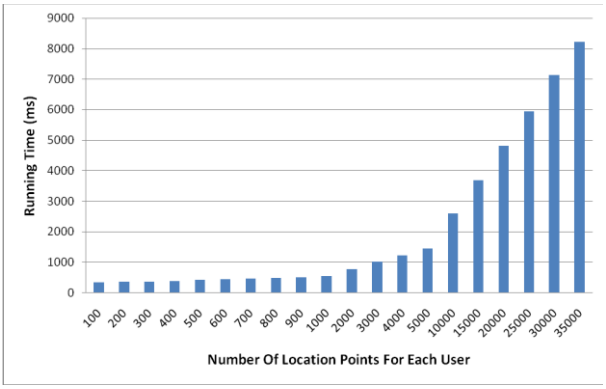


Figure 12. Running Time for People-to-People Affinity Learning Algorithm for Two Users

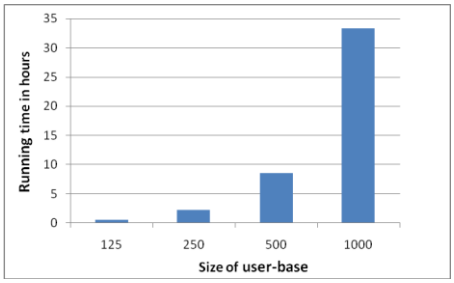


Figure 13. Running Time of GPI Function of Number of Users

To evaluate the runtime cost of GPI, the algorithm implemented in the People-Place module, we generated data for 1000 users and 100 places. Each user visits 8 random places per day for one hour each, and the location is updated every 10 seconds. Although GPI is unique in its ability to identify previously unknown social groups and their associated places, Figure 13 shows that it does not scale well with the number of users. This algorithm incorporates two major computationally expensive steps. The first step involves the analysis of location traces for place extraction for each individual user, and it has the same cost for each user. The second step involves a comparison between the places visited by all the users to determine user co-presence and subsequently detect groups. Asymptotically, the cost of this step is the $O(N^2)$.

In order to improve the running time of GPI as well as other algorithms that compare mobility traces of large user populations, we are working on a new technique that avoids checking each member of the population against the entire population. This technique stems from a similar problem encountered in computer graphics when tracing the path of a simulated light-ray during an image rendering process known as ray tracing. The naïve approach checks the light-ray against every object in the scene to determine intersection. To overcome this complexity, the ray tracer first bounds all the objects in easily computed bounding boxes so that the light-ray is only compared against objects its path intersects. Similarly, if a small amount of pre-computation is performed, it is possible to divide the user population into smaller bounding regions, and only compare a particular person against the bounding regions surrounding his or her location. Our initial results using this method have been extremely promising; once the method is incorporated in MobiSoC, it should significantly improve the running times of location-based learning algorithms.

6.2 Smart Phone Experiments

Since most MSCAs use location information or benefit from location-based patterns identified by MobiSoC, we first tested the accuracy of our location engine. We used the centroid and fingerprinting methods described in Section 5. Table 2 shows results for the building that hosts our department. Since this building has a good density of access points, we obtained very good

accuracy for our purposes (room level accuracy). As expected, fingerprinting produced better results with the laptop and HTC Wizard phone. However, the lesson learned while analyzing these results was that device heterogeneity significantly impacts the accuracy. For instance, the iMate KJam phone constantly had the worst results. Additionally, the centroid method performed better than fingerprinting for this phone; the cause is due to recording the fingerprints with the HTC phone. Finally, experiments around the entire campus (fully WiFi covered) proved that an accuracy of 10-15m is achievable almost everywhere.

With the location engine running on the phone, the users can decide when and where to share location with the middleware. However, when it runs, the location engine might consume a significant amount of battery power. This is especially true for situations when the location detection (scanning for WiFi access points and computation) and update must be done frequently to benefit from real-time location-awareness. Therefore, the next experiment quantifies the impact of the location engine on the smart phone battery life using the centroid method. This experiment used the iMate KJam phone, whose battery lifetime when the WiFi interface is continuously on is 8 hours. The acbTaskMan [41] was used to record the current draw on the phone's battery.

	Laptop	HTC Phone	i-Mate Phone
Average Centroid Accuracy (m)	4.77	3.71	7.55
Average Fingerprint Accuracy (m)	2.56	2.84	8.25
Average Number of Visible Access Points	5.5	6.8	4.8

Table 2. Location Accuracy on Three Mobile Devices

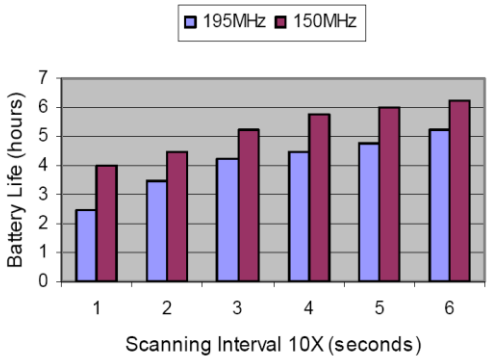


Figure 14. Battery Lifetime as Function of Location Detection Period

Figure 14 shows the battery lifetime as function of location detection period. We also plot the battery behavior when under-clocking the CPU at 155 MHz instead of its regular 195 MHz. In absolute terms, if we consider a student on campus attending a 3-hour class and spending some time with her friends after class, the results demonstrate that it is feasible to run applications on the phone. However, the battery lifetime is reduced by at least 25% by running the location engine every minute or less, and this is a significant issue. One way to save battery power is to optimize the frequency of location detection and transmission. Another way is to turn off the WiFi interface when location is not transmitted for a longer period of time; since our applications are event-based and receive events when they transmit location data, this optimization will not impact their functionality. Finally, we note that newer phones have significantly longer battery lifetimes [19].

We implemented several variations of the first optimization mentioned above. The next experiments compare these methods against the baseline, namely *Method A*, which computes and sends the location periodically, with a period of 10s. *Method B* computes location as often as Method A, but applies a threshold to determine if the new location represents a significant deviation from the previously transmitted location. Method B is expected to save CPU cycles and

energy by sending fewer location updates. Two thresholds are used: a *room* level threshold of 12m, and a *building* level threshold of 60m. Additionally, if a location update is not transmitted after 5 minutes, the most recent location is transmitted regardless of its significance. This is to preserve a certain level of contact between the phone and MobiSoC. *Method C* applies a significance threshold identical to Method B, but it also adapts its location detection period. In this way, it can save more CPU cycles and energy than method B. Each time the deviation of a detected location is insignificant, the detection period is increased. The increase factor is 1.5, and the starting period is 10s. The period is reset to its initial value upon a significant location deviation and the subsequent location transmission.

This experiment used a trace of a campus tour with a walking speed of 4 km/h and several stops of varying length at different campus locations. The overhead to determine the next location in the trace is 20ms. This is negligible, considering the location detection takes between 1/2s and 1s. The phone was placed in an access point rich location, and each test was run for 1.5 hours. For more realistic experiments, we injected "jitter" of 10, 15, and 20 meters in the computed location. Instead of measuring directly the battery power, we compare these methods in terms of the individual components that impact the energy consumption: the processing time, the number of recordings (i.e., how many times the location is computed), and the number of updates transmitted to MobiSoC.

Figure 15 shows the percentage of the total execution time spent detecting location, transmitting locations, and other tasks (such as comparing location distances). This metric quantifies the location engine overhead (i.e., the remaining percentage can be allocated to applications). The results show that methods B and C reduce the execution time by at least 30% and at most 8 times compared to method A. The first cause for this significant improvement is the reduction in the time spent transmitting location in B and C to at least half the time spent by A. Methods A and B spend roughly equivalent percentages of time calculating location due to their utilization of the same fixed-period location detection. Method C, however, reduces the location detection time to at least half this value. As expected, when using a building threshold, we observe even more significant savings.

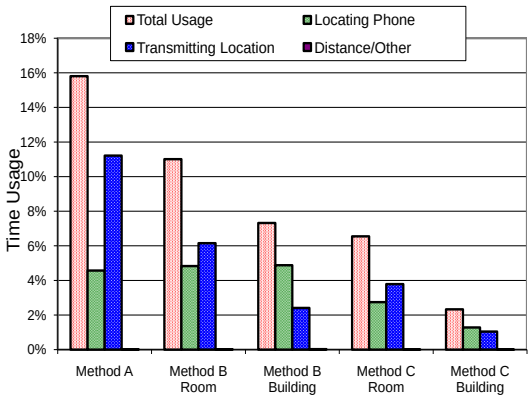


Figure 15. Time Usage for Our Localization Methods

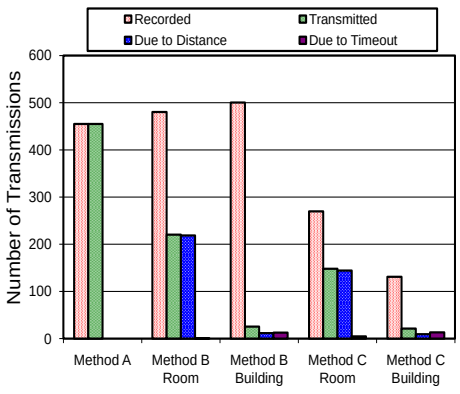


Figure 16. Location Recording vs. Location Transmission

We also see that method C spends about the same amount of time detecting location as it spends transmitting. This result is explained by Figure 16, which shows that method C, especially

C Room, not only records fewer location estimates, but also transmits a higher percentage of the recorded locations. As we can see in the same figure, most of the transmissions are triggered by distances larger than the threshold, with very few being triggered by a timeout. Unexpectedly, this figure shows that B records more points than A. The cause for this result is the reduction in execution time overhead associated with method B, which allows for more recordings. The final conclusion of this experiment is that method C is indeed the most efficient, and it results in a good representation of the real user path (this result in not shown for the sake of brevity).

Besides the location engine, another factor that could consume significant resources on the phones is security, specifically cryptographic primitives. To this end, we investigated two symmetric encryption methods, 3DES and AES, and one asymmetric encryption method, RSA, for their expected run-time impact on the phones. 3DES and AES were tested once without encryption performed on their private keys and once with RSA applied to their private keys. This was done to analyze the common hybrid method of using a more robust encryption algorithm to secure a private key embedded in a header, which is used to decrypt the main body of a transmission.

Figure 17 presents the running time of these methods (note the logarithmic scale) for two packet sizes: 20 byes and 1542 bytes. The first size corresponds to SHA1 hash length. The second corresponds to the size of a SOAP encoded location update, but it is representative for many short messages used by MSCAs. We believe that many times only message authentication and integrity will be required, but not message confidentiality. For instance, this might be enough for location updates considering that providing confidentiality for every location update could be quite expensive in terms of CPU cycles and battery power. Therefore, this experiment evaluated: (*i*) the cost of digitally signing and verifying the hash of the 1542-byte message (i.e., integrity and authentication); (*ii*) encrypting/decrypting the entire message (i.e., confidentiality).

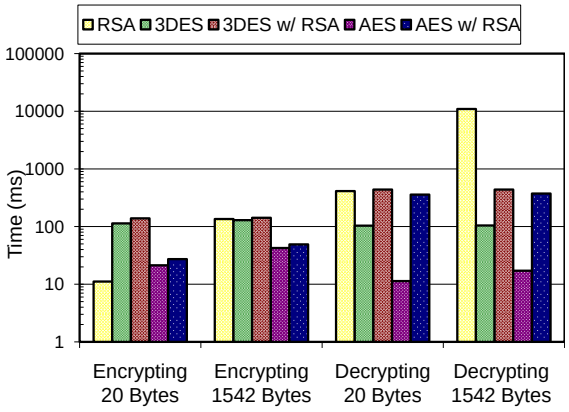


Figure 17. Running Time for Encryption and Decryption Algorithms

Although we evaluated both SHA1 and MD5, the graph shows the results just for the former as the overall performance was very similar. The time to compute the hash is not included (the values were 2.3ms for MD5 and 2.7ms for SHA1). The results demonstrate that RSA is the best solution to digitally sign message digests (i.e., hashes) on the phone. However, RSA is among the most expensive to verify the digital signature. Nevertheless, it can be used for location updates, which are signed on the phones and verified at the server side. For providing confidentiality of

larger messages (even the 1542-byte in our case), AES with or without RSA applied to its cipher key (function of the desired security) yields the best performance. Finally, the absolute numbers show that these choices are feasible on the phones.

6.3 User Study

We performed a user study of Clarissa to understand the social usefulness of MSCAs. In this study, we also investigated the benefits of location and geo-social patterns on social matching as well as the potential impact of privacy concerns on mobile social computing. This study was conducted on the NJIT campus in April 2008 with more than 50 participants over three weeks. The majority of the participants were undergraduate students at NJIT, with a relatively even population of males and females. To be able to assess the effect of location and geo-social patterns, we chose to have both mobile and on-line users. The 13 mobile users were provided with HTC Wizard smart phones running the Clarissa client. The demographics of these users was: 7 males, 6 females; 7 undergraduate, 6 graduate; 6 on-campus, 7 commuters; all full time students; average age was 25. The on-line users were provided with our Facebook version of Clarissa [42]. While mobile users were allowed to use the Facebook version to register and explore matches, they were encouraged to use the phone version. We collected and analyzed detailed logs of per-user application usage. At the end of the study, they were required to fill out an evaluation form, and interviews were conducted after the evaluation forms were collected and analyzed.

	User Rating	**Variance**
Data about existing friends	2.83	1.06
Meeting new people	3.41	0.81
Discovering activity partners	3.83	0.87

Table 3. Clarissa's Average User Perceived Usefulness

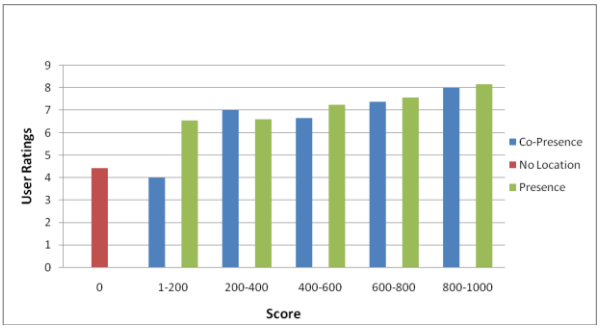


Figure 18. Effect of Geo-Social Patterns on Match Ratings

Table 3 shows Clarissa's average user perceived usefulness, rated from 5 (extremely useful) to 1 (not at all). The results demonstrate that on average the users found this application useful or very useful (rating 4) for meeting new people and discovering activity partners. Since the application was not designed specifically for sharing information with friends (and considering that other alternatives exist for that), we observe a lower rating for this part. Nevertheless, we found the overall results very encouraging for future MSCAs.

Figure 18 presents the effect of two geo-social patterns on the quality of matches. The users rated all their matches with scores from 1 (worst) to 10 (best). The two geo-social patterns are *presence* and *co-presence*. Presence captures how much time two users spend at common places without requiring simultaneous presence (e.g., one user goes to the gym in the morning and another goes in the evening). Co-presence requires simultaneous presence at a place. In MobiSoC, the affinity between two users along each pattern is represented as a score ranging from 1 (worst)

to 1000 (best). For comparison purposes, the graph also shows the average rating for matches computed without location information (i.e., for on-line users only). The results demonstrate a very significant effect of geo-social patterns on average rating scores. For example, the score for mobile matches with very high affinities along these patterns are over 90% higher than the score for on-line matches. In general, we observe that even a moderate geo-social affinity produces results at least 50% better. Finally, since presence includes co-presence, it ends up with slightly higher scores. These results confirm the expectation that geography, and especially visiting common places, strengthens social relationships.

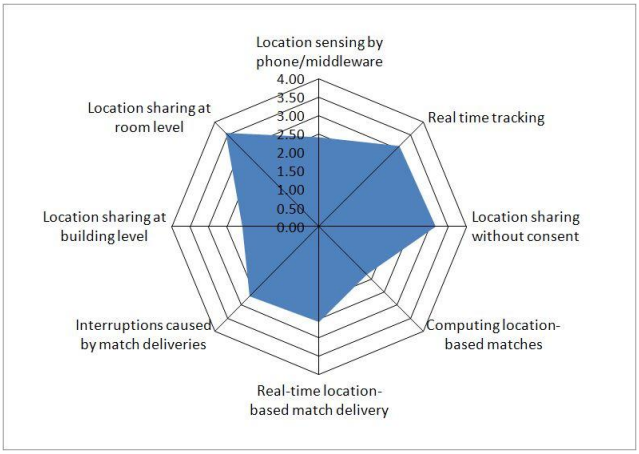


Figure 19. Ratings of Privacy Concerns

Figure 19 shows users' answers to questions about privacy concerns. As part of the Clarissa evaluation, users were asked to rate their privacy concerns on a scale from 1 (not at all concerned) to 5 (extremely concerned). The questions are listed on the axes, and the intersections of axes with the shaded area mark the average response. The results show that users are minimally threatened by the use of their location data in computing matching scores. In terms of sharing location with other users, we observe that most users accept sharing at a building level, but strongly oppose sharing at a room level. We received mixed responses concerning co-location based events invading privacy (i.e., a user might see the match around) and alerts causing interruptions. In terms of real-time location tracking, users are normally concerned for the general case, but they are significantly less concerned once they understand that they have control of location sharing on the phone and the middleware enforces privacy policies.

7. Conclusions

This article presented MobiSoC, a middleware that provides a common platform for rapid development and deployment of mobile social computing applications. This middleware captures the social state of physical communities, learns previously unknown patterns from the emergent geo-social data, and augments the social state with this new knowledge. Additionally, it provides mechanisms to share social state data in real-time with applications running on mobile devices, while respecting user's privacy concerns. We implemented MobiSoC as a flexible service oriented architecture and used it to build Tranzact and Clarissa, two prototype applications running on smart phones. The experimental results showed that MobiSoC can provide reasonable

performance, and data caching can further improve this performance. We also demonstrated that adaptive localization on the phones works well in terms of accuracy and resource consumption, and we proved the feasibility of certain levels of security on the phones. The user study for Clarissa showed that people consider such applications useful and are willing to share their location to benefit from them. The same study demonstrated that location and geo-social patterns can greatly improve the social quality of MSCAs.

8. Acknowledgements

This material is based upon work supported by the National Science Foundation under Grants No. CNS-0454081, IIS-0534520, CNS-0520033, and CNS-0831753. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. The authors thank Quentin Jones for his participation in the initial design of MobiSoC. We also thank Sanil Paul, Arjun Anand, and Jason Wolf for help with the experiments on smart phones.

References

[1] Q. Jones, S. Grandhi, S. Karam, S. Whittaker, C. Zhou, and L. Terveen. Geographic place and community information preferences. In *Journal of Computer Supported Cooperative Work*. 2007.

[2] Q. Jones, S. Grandhi, L. Terveen, and S. Whittaker. People-to-people-to-geographical places: The P3 framework for location-based community systems. In *Journal of Computer Supported Cooperative Work*. Kluwer Academic Publishers, 2004.

[3] Nokia Sensor. http://www.nokia.com/sensor/.

[4] M. Terry, E. D. Mynatt, K. Ryall, and D. Leigh. Social Net: Using Patterns of Physical Proximity Over Time to Infer Shared Interests. In *Proc. Human Factors in Computing Systems (CHI 2002)*, pages 816-817, 2002.

[5] N. Eagle and A. Pentland. Social serendipity: mobilizing social software. In *Pervasive Computing, IEEE*, volume 4(2), pages 28-34. 2005.

[6] iTunes Starbucks. http://www.apple.com/itunes/starbucks/.

[7] Geo-notes: Place based virtual notes. http://csd.ssvl.kth.se/csd2002-geonotes/.

[8] DodgeBall. http://www.dodgeball.com/.

[9] VZ Navigator. https://vznavigator.vzw.com/.

[10] Bottazzi, D., Montanari, R., Toninelli, A. (2007). Context-Aware Middleware for Anytime, Anywhere Social Networks. IEEE Intelligent Systems, 22(5), 23-32.

[11] W. G. Griswold, R. Boyer, S. W. Brown, and T. M. Truong. A Component Architecture for an Extensible, Highly Integrated Context-Aware Computing Infrastructure. In *International Conference on Software Engineering (ICSE 2003)*, pages 363-372, May 2003.

[12] Bicocchi, N., Castelli, G., Mamei, M., Rosi, A., Zambonelli, F. Supporting Location-Aware Services for Mobile Users with the Whereabouts Diary. In *Proceedings of the 1st International Conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications (MOBILWARE'08)*, pages 1-6, February 2008.

[13] Barabasi, A. L. (2002). Linked: The New Science of Networks. Perseus Publishing.

[14] Newman, M. The Structure and Function of Complex Networks. SIAM Review, 45(2), 167-256.

[15] Iamnitchi, A., Foster, I. Interest-Aware Information Dissemination in Small-World Communities. In *Proceedings of High Performance Distributed Computing (HPDC)*, pages 167-175, July 2005.

[16] Miklas, A. G., Gollu, K. K., Chan2, K. K.W., Saroiu, S., Gummadi, K. P., De Lara, E. Exploiting Social Interactions in Mobile Systems. In *Proceedings of the 9th International Conference on Ubiquitous Computing (UbiComp'07)*, pages 409-428, September 2007.

[17] Campbell, A. T., Eisenman, S. B., Lane, N. D., Miluzzo, E., Peterson, R. A., Lu, H., Zheng, X., Musolesi, M., Fodor, K., Ahn, G. (2008). The Rise of People-Centric Sensing. IEEE Internet Computing, 12(4), 12-21.

[18] Burke, J., Estrin, D., Hansen, M., Parker, A., Ramanathan, N., Reddy, S., Srivastava, M. B., Participatory Sensing. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys'06)*, October 2006.

[19] Gaonkar, S., Li, J., Choudhury, R. R., Cox, L., Schmidt, A. Micro-Blog: Sharing and Querying Content Through Mobile Phones and Social Participation. In *Proceedings of the Sixth International Conference on Mobile Systems, Applications, and Services (MobiSys'08)*, pages 174-186, June 2008.

[20] Froehlich, J., Chen, M.Y., Consolvo, S., Harrison, B., Landay, J. A. MyExperience: A System for In Situ Tracing and Capturing of User Feedback on Mobile Phones. In *Proceedings of the Fifth International Conference on Mobile Systems, Applications and Services (MobiSys'07)*, pages 57-70, June 2007.

[21] Hengartner, U., Steenkiste, P. (2006) Avoiding Privacy Violations Caused by Context-Sensitive Services. Pervasive and Mobile Computing. PerCom 2006 Special Issue, 2(4), 427- 452.

[22] Hong, J., Landay, J. An Architecture for Privacy-sensitive Ubiquitous Computing. In *Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services (2004)*, pages 177-189, June 2004.

[23] Cox, L. P., Dalton, A., Marupadi, V. SmokeScreen: Flexible Privacy Controls for Presence-Sharing. In *Proceedings of the Fifth International Conference on Mobile Systems, Applications, and Services (MobiSys'07)*, pages 233-245, June 2007.

[24] Cornelius, C., Kapadia, A., Kotz, D., Peebles, D., Shin, M., Triandopoulos, N. Anonysense: Privacy-Aware People-Centric Sensing. In *Proceedings of the Sixth International Conference on Mobile Systems, Applications, and Services (MobiSys'08)*, pages 211-224, June 2008.

[25] Dey, A. K., Salber, D., Abowd., G. (2001). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Human-Computer Interaction, 16(2), 97-166.

[26] Roman, M., Hess, C., Cerqueira, R., Campbell, R. H., Nahrstedt, K. (2002). Gaia: A middleware infrastructure to enable active spaces. IEEE Pervasive Computing Magazine, 4(1), 74-83.

[27] Grimm, R., Davis, J., Lemar, E., MacBeth, A., Swanson, S., Anderson, T., Bershad, B., Borriello, G., Gribble, S., Wetherall, D. (2004) System support for pervasive applications. ACM Transactions on Computer Systems, 22(4), 421-486.

[28] Riva, R., Nadeem, T., Borcea, C., and Iftode, L. Context-aware Migratory Services in Ad Hoc Networks. In IEEE *Transactions on Mobile Computing*, volume 6(12), pages 1313-1328. December 2007.

[29] Bellavista, P., Corradi, A., and Foschini, L. MUM: A Middleware for the Provisioning of Continuous Services to Mobile Users. In *Proceedings of the 9th IEEE International Symposium on Computers and Communications (ISCC'04)*. IEEE Computer Society Press, Jun 2004.

[30] Wang, L., and Jajodia, S., and Wijesekera, D. Securing OLAP data cubes against privacy breaches. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 161-175. May, 2004.

[31] Weme: WebSphere Everyplace Micro Edition. http://www.ibm.com/software/wireless/weme/.

[32] Xstream: JAVA-XML parser. http://xstream.codehaus.org/.

[33] Ksoap: Simple object access protocol for J2ME. http://ksoap.objectweb.org/.

[34] Schilit, B., LaMarca, A., Borriello, G., Griswold, W., McDonald, D., Lazowska, E., Balachandran, A., Hong, J., and Iverson, V. Challenge: Ubiquitous Location-Aware Computing and the Place Lab Initiative. In *Proceedings of the 1st ACM International Workshop on Wireless Mobile Applications and Services on WLAN (WMASH 2003)*, San Diego, CA, Sep 2003.

[35] Cheng, Y., Chawathe, Y., LaMarca, A., and Krumm, J. Accuracy Characterization for Metropolitan-scale Wi-Fi Localization. In *Proceedings of Mobisys 2004*, Washington, DC, 2004.

[36] Gupta, A., Paul, S., Jones, Q., and Borcea, C. Automatic Identification of Informal Social Groups and Places for Geo-Social Recommendations. In *The International Journal of Mobile Network Design and Innovation (IJMNDI)*, 2(3/4), 159-171, 2007.

[37] Hightower, J., Consolvo, S., LaMarca, A., Smith, I., Hughes, J. and Borriello, G. (2005). Learning and Recognizing the Places We Go. In *Proceedings of International Conference on Ubiquitous Computing (2005)*, Tokyo, Japan, September, pp.159–176.

[38] Kang, J.H., Welbourne, W., Stewart, B. and Borriello, G. Extracting Places from Traces of Location. In *Proceedings of the International Workshop on Wireless Mobile Applications and Services on WLAN (2004)*, Philadelphia, USA, October, pp.110–118.

[39] Dunbar, R. (1992) Neocortex size as a constraint on group size in primates, Journal of Human Evolution 22: 469-493.

[40] Facebook MemcacheD. http://developers.facebook.com/opensource.php.

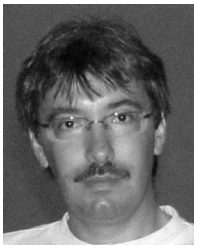
[41] acbTaskMan. http://www.acbpocketsoft.com/Products/acbTaskMan/acbTaskMan-Overview-7.html.

[42] Clarissa Application on Facebook. http://apps.facebook.com/matching/.

Ankur Gupta is a PhD candidate in the Department of Computer Science at the New Jersey Institute of Technology (NJIT). His research interests include mobile and ubiquitous computing, middleware, and algorithms. He received his MS in computer science from NJIT in 2005.
Email: ag59@njit.edu

Achir Kalra is a PhD candidate in the Department of Computer Science at NJIT. His research interests include mobile social computing and social matching systems. He received his MS in computer science from NJIT in 2005.
Email: ak95@njit.edu

Daniel Boston is a PhD student in the Department of Computer Science at NJIT. His research interests include networking, mobile social computing, artificial intelligence, security systems, and algorithms. He received his BS in Computer Science from NJIT in 2008.
Email: djb38@njit.edu



Cristian Borcea is an assistant professor in the Department of Computer Science at NJIT. His research interests include mobile computing, middleware, ad hoc networks, and distributed systems. He received his PhD in computer science from Rutgers University in 2004. He is a member of the IEEE, ACM, and Usenix.
Email: borcea@cs.njit.edu