The MobiSoC Middleware for Mobile Social Computing: Challenges, Design, and Early Experiences

(Invited Paper)

Cristian Borcea
Dept. of Computer Science
NJIT
Newark, NJ 07102, USA
borcea@cs.njit.edu

Ankur Gupta
Dept. of Computer Science
NJIT
Newark, NJ 07102, USA
ag59@njit.edu

Achir Kalra
Dept. of Computer Science
NJIT
Newark, NJ 07102, USA
ak95@njit.edu

Quentin Jones
Dept. Of Information Systems
NJIT
Newark, NJ 07102, USA
qjones@njit.edu

Liviu Iftode
Dept. Of Computer Science
Rutgers University
Piscataway, NJ 08854, USA
iftode@cs.rutgers.edu

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 paper presents MobiSoC, a middleware that enables MSCAs 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.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed Applications; D.2.11 [**Software Engineering**]: Software Architectures; H.4.3 [**Information Systems Applications**]: Communications Applications

General Terms

Design, Experimentation, Human Factors, Algorithms

Keywords

Mobile social computing, middleware, smart phones

Mobilware'08 February 12-15, 2008, Innsbruck, Austria

1. INTRODUCTION

Social computing applications such as Facebook [1], MySpace [2], and LinkedIn [3] 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 interpersonal 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 reconnect with their physical communities and surroundings. For instance, MSCAs can answer questions such as: 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? With research showing that users are increasingly willing to share their profile information and location in return for services, the time is ripe to develop MSCAs that can answer queries and provide recommendations about people, places, and events of interests anytime, anywhere [4].

This paper presents MobiSoC, a mobile social computing middleware that provides support for programming and deploying MSCAs. 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 [5]. 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 de-

veloping 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 lightweight versions of JAVA on mobile devices. To develop MSCA applications, it is essential to collect accurate and continuous user location data both indoors and outdoors. Furthermore, the location system has to be cheap and easily deployable. Finally, it has to allow users to control the sharing of location data for privacy reasons. Considering all these requirements, we chose Intel's PlaceLab location engine that computes location on mobile devices using the position and signal strength of visible WiFi access points. In our campus, we have at least three visible access points almost everywhere, and consequently, we obtained an accuracy of 10 - 15 meters. We used MobiSoC to build two MSCAs: Clarissa, a location-based mobile social matching application, and Tranzact, an application for place-based ad hoc social collaboration. These applications were tested successfully on Windows-based smart phones, which connect to the Internet over WiFi.

The rest of this paper is organized as follows. Section 2 presents an MSCA taxonomy and motivating examples. Section 3 discusses the main challenges that face MSCA development. Section 4 describes the MobiSoC design and explains its core modules. Section 5 explains how to build applications over MobiSoC and illustrates that with two prototype applications. Section 6 contains implementation details and early experiences. We discuss related work in Section 7 and conclude in Section 8.

2. APPLICATIONS

MSCAs can be categorized into people-centric and place-centric. This section illustrates each category with several examples. We assume that users carry mobile locatable devices, which can access services across the Internet using either short range wireless interfaces (WiFi, Bluetooth) or cellular interfaces. Furthermore, all these examples assume that mobile users are willing to share their profile information (including location) with trusted applications under certain privacy constraints.

2.1 People-Centric Applications

Mobile Social Matching. This application leverages geo-social patterns to provide enhanced social matching recommendations. Users initiate the matching process by registering a match request with the application, which holds details about the desired match. For instance, a student can specify that she looks to hang out with a person with similar interests on Monday from 2PM to 6PM. The application uses real-time and historical location information, the social network graph, and the basic user profiles to compute affinities between potential matches. A match alert is delivered to the mobile device when all the constraints for the match are satisfied. In our example, the application can select as best match a nearby student who shares common friends (known from the social network graph) and common interests (known from the location history analysis) with the requester.

Social Event Planner. This application can be used to plan and generate invitation lists for events, such as seminars or outdoor activities. For example, if there is a seminar on Mobile Computing in the CS department building, the application can suggest people who should be invited to the seminar. The list includes department's members who have explicitly listed Mobile Computing as an area of interests in their profiles, but also people who visit frequently the networking and systems labs. Additionally, the social networks of these people are searched for friends from other departments that might be interested in the topic.

2.2 Place-Centric Applications

Ad Hoc Social Collaboration. This application submits location-based queries to mobile users identified by certain profile properties. For example, let us consider that a user sitting in his office might wish to know the current menu at the cafeteria (which is not posted anywhere except inside the cafeteria). Since the user does not know who is in the cafeteria at that moment, the application helps her by identifying members of her social networks which are there and asks them to answer the query. This application can also be used to deliver queries anonymously to any person who can provide information about a certain place subject to privacy and reputation constraints.

Place Information. This application associates social semantics to a place based on the profiles and interests of the users who visit that place. Then, it could recommend the place to users with similar interests or answer queries about it. For instance, a CS student looking for a place to hang out could be recommended to visit the game room of the student center on Tuesday evenings, when it is typically occupied by CS students. Using the same application, the campus administration can discover places which need improvement by checking the statistical information about places (e.g., type, size, and demographics of the groups that meet at each place). For instance, the settings and ambiance in certain rooms of the student center can be modified according to the number of students who spend time there.

3. CHALLENGES

Social computing applications in the Internet, such as MySpace [2] and Facebook [1], 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, the software platforms for mobile social computing must address the following challenges.

3.1 Social Data Collection and Management

MSCAs require complex geo-social community data about people and places. Static data such as people demographics or places' coordinates are easy to collect, manage, and validate. Dynamic data, on the other hand, are much harder to handle. For instance, a software platform supporting MSCAs must provide mechanisms to: (1) capture ties between users and between users and places, (2) model, validate, and store these ties, and (3) effectively share community data among multiple applications.

3.2 Location Collection and Management

Most MSCAs need users' location data to function properly. Therefore, it is essential to provide infrastructure support to collect real-time user location. Ideally, a location system should be accurate, scalable, cost effective, easily deployable, work both indoors and outdoors, and allow users to control the sharing of their location. No currently available system has all these features, and in fact, some of the features are contradictory. Therefore, system designers will have to consider which are the most important features for MSCAs when choosing the location system. Furthermore, the software that collects the location and shares it with MSCAs must scale to a large number of concurrent location updates to accommodate a large user population and the need of MSCAs to react quickly to location changes.

3.3 Learning Emergent Geo-Social Patterns

Communities have rich geo-social ties, which cannot be captured directly from the user profiles. However, the collection of raw data about people and places over time could be used to determine emergent patterns which are not apparent otherwise. 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. In order to identify such patterns, we need to model the global state of a community and design algorithms which can extract new knowledge out of this state.

3.4 Privacy

In addition to the privacy issues for desktop-based social computing applications, the software platform which supports MSCAs must also cope with location privacy, which is highly sensitive for mobile users. The MSCAs and their software platform must ensure that users cannot track each other. An even more difficult privacy problem than tracking is the inference of social ties based on location. For instance, if Bob is allowed to see John's location, and he sees that John is too often located at Alice's office, he might infer that they might have a romantic relationship. While solutions exist for direct inference channels (e.g., database query evaluation), more research is necessary to prevent indirect inference channels such as the one in this example.

3.5 Energy Efficiency

Battery power represents the most important physical limitation faced by MSCAs developers. For example, the basic question asked by a user who plans to run such an application on her smart phone is: How long can I use my phone if I run this application? Therefore, developers need to understand the power consumption characteristics of each component of the mobile devices in order to design energy efficient applications and protocols. 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, common computationally intensive tasks required by multiple applications or multiple users should be offloaded to servers. Furthermore, data aggregation methods should be designed to run at the servers in order to reduce both the computation and communication costs.

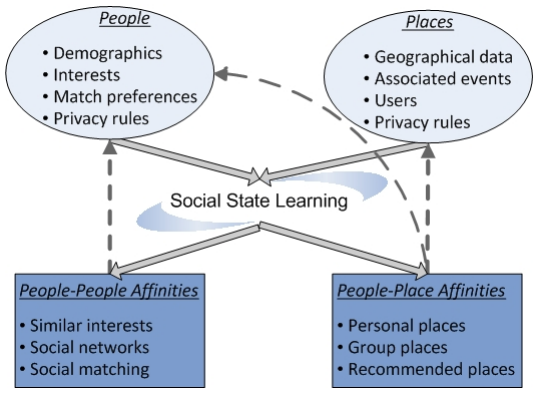

Figure 1: Social State of a Community

4. MOBISOC ARCHITECTURE

This section presents the design of MobiSoC, our mobile social computing middleware designed to address the challenges discussed in the previous section. 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.

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. Additionally, having servers in the system architecture helps to improve the 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 for thousands of mobile clients. In the following, we present each of the middleware modules.

4.1 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 also provides 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 deter-

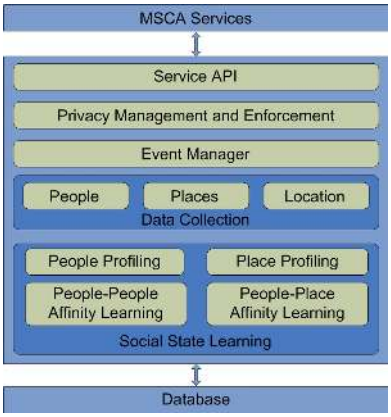

Figure 2: MobiSoC Architecture

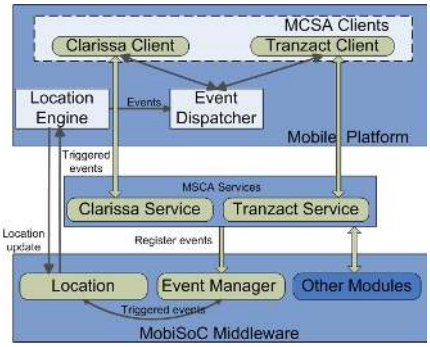

Figure 3: Building Applications over MobiSoC

mine 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.

4.2 Social State Learning

This module learns emergent social state information. 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. Ad hoc social groups can be discovered based on co-location in the same place at the same time. 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 basically performs temporal synchronization on the mobility traces and uses clustering techniques to determine repeated user co-presence at a place.

4.3 Event Manager

This module is used for asynchronous communication with the applications. The applications 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. 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 change frequently 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 savings 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 poll periodically the middleware for event notifications. To reduce useless communication which could impact the battery lifetime, our implementation, as illustrated in Section 5, takes advantage of the location engine running on mobile devices to retrieve event notifications during the location updates.

4.4 Privacy Management and Enforcement

This module manages and enforces privacy rules on behalf of the entities in the system (users and applications). Entities register their privacy preferences using a privacy statement, which has a primary entity that issues the statement and a secondary entity on which the statement applies. A privacy statement 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. Currently, we support 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.

5. 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

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

Figure 4: Middleware API

and Clarissa, successfully developed and tested on smart phones.

5.1 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.

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.

5.2 API and Code Examples

The current API exposed to MSCA services is presented in Figure 4. 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.

Tranzact is an application for place-based ad hoc social collaboration. Its clients send queries for real-time information from various places. Figure 5 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 that 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 verifies if the potential destinations are willing to accept events from this application. In this case, the privacy constraints, verified through the Privacy Management module, are set by a user for herself (i.e., when and where she wants to receive Tranzact events). The available users receive the request using our event-based communication mechanism. Responses are sent back through the same mechanism.

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

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

Figure 5: Tranzact pseudo-code

Clarissa is a location-based mobile social matching application. Figure 6 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. Then, it 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 requester, which are triggered by the co-presence with potential matches during the specified time interval.

```
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 6: Clarissa pseudo-code

6. IMPLEMENTATION PROTOTYPE AND EARLY EXPERIENCES

We built a prototype implementation for MobiSoC using a service-oriented architecture, which supports evolution by providing modularity, extensibility, and language independence. 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 TyTN* smart phones. WebSphere Everyplace Micro Edition Java (WEME) [6] was selected as the Java Micro Edition implementation because it runs reliably on our smart phones with ARM processors and Window Mobile operating system. Personal Profile 1.0 and MDIP 2.0 is the minimum configuration required.

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 to serialize Java objects and send them over sockets, but this option does not provide for language independence. The next option considered was to use a lightweight parser [7] 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 KSOAP J2ME [8] 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 provides fast performance comparable to XML-RPC, yet provides support for custom data types.

The location engine on the clients is a modified version of Intel's Place Lab software [9]. This engine estimates the user location based on the location and the signal strength of the WiFi access points visible from the mobile device. Mobile devices hold a database consisting of access point names and their locations in the area of interest. 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, such as the one described in [10], to improve the location accuracy. Currently our system provides room level accuracy (10-15 meters). However, a significant problem that we observed is that accuracy differs significantly (up to 10 meters) for different mobile devices. In the near future, we plan to investigate solutions that accommodate device heterogeneity.

While this location engine provides many benefits, we had concerns about its power consumption on mobile devices. To quantify the effect of the location engine on the overall energy consumption on smart phones, we ran experiments in which the location was computed and delivered over WiFi at intervals between 10 seconds and 1 minute [11]. For these experiments, we used the centroid version of the location calculation. The results showed that the battery lifetime lasted between 4 and 6 hours, which translates into a consumption of 25%-50% of the total battery power. The main conclusion is that adaptive strategies are necessary to compute and deliver the location in real-time in order to reduce the energy consumption. For instance, the location can be updated only when users move more than a certain threshold. Furthermore, social knowledge can be used to decrease the frequency of the updates or even to turn off the engine for certain periods of time (e.g., attending weekly meetings or classes).

While the People-to-People Affinity module of the middleware is relatively straightforward to implement, the People-to-Places Affinity module is more difficult due to the additional uncertainties added by the location errors. The first algorithm which we developed for this module is GPI, an algorithm which learns previously unknown ad hoc social groups and their meeting places [12]. The middleware can share the results of GPI with geo-social recommendation applications subject to privacy constraints. For instance, new students can learn about popular hangouts on campus or faculty could learn about students attending research seminars on certain topics. Our 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%. 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.

7. RELATED WORK

A number of projects developed toolkits and middleware architectures for mobile context-aware applications. Aura [13] is a middleware architecture that allows mobile applications to adapt to dynamically-changing resources. GAIA [14] provides dynamic resource discovery in ubiquitous computing environments and system configuration according to the available resources. One.world [15] provides a framework for developing pervasive computing applications with a focus on resource discovery and migration to cope with changing usage environments. The context toolkit [16] presents a conceptual framework that supports the acquisition, representation, delivery, and reaction to context information.

Context-aware migratory services [17] is a distributed programming framework that provides a client-server model in mobile ad hoc networks (MANETs), where services migrate to different nodes to maintain a semantically correct and continuous interaction with clients. A similar idea in a different context is presented in the MUM middleware [18], which aims at relieving mobile services from the burden of maintaining service continuity in dynamic environments, with multimedia services moving among different wireless access points. REDMAN [19] is a middleware that disseminates, manages, and retrieve replicas of resources of common interest made available by cooperating nodes in dense MANETs. Spatial Programming [20] is a runtime system that provides an imperative programming model for distributed applications in MANETs using location and properties to name the nodes of interest. Contory [21] is a middleware for context provisioning on smart phones that provides an SQL-like interface to query context data across the Internet or MANETs.

A number of mobile social applications have been previously developed. Lovegetty [22] is a wireless-enabled, spontaneous matchmaking device that works within its limited radio transmission range. Social Net [23] infers shared interests between people by storing the IDs of the nearby devices and analyzing this co-location data collected over a long period of time. Social Serendipity [24] is another similar mobile phone-based application using Bluetooth and a database of user profiles to recommend face to face interactions between nearby users who share common preferences. GeoNotes [25] allows users to post virtual notes at places, which can be read by other users visiting the same place. Context-aware recommender systems which can assist users with shopping are presented in [26, 27]. The Active Campus [28] project is also host to several social computing applications built on top of a centralized client-server architecture.

8. CONCLUSIONS

This paper 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. MobiSoC was developed as part of the SmartCampus project [29], which investigates the benefits of mobile social computing in university campus settings. As this project will provide several hundred students with smart phones, we will be able to test our middleware and applications in a real-life large scale community.

9. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grants No. CNS-0454081, IIS-0534520, CNS-0520033, and CNS-0520123. 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.

10. REFERENCES

[1] Facebook (2004). http://www.facebook.com.

[2] MySpace (2003). http://www.mysace.com.

[3] LinkedIn (2002). http://www.linkedin.com/.

[4] 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.

[5] 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.

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

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

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

[9] B. Schilit, A. LaMarca, G. Borriello, W. Griswold, D. McDonald, E. Lazowska, A. Balachandran, J. Hong, and V. Iverson. 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.

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

[11] A. Anand, C. Manikopoulos, Q. Jones, and C. Borcea. A Quantitative Analysis of Power Consumption for Location-Aware Applications on Smart Phones. In *Proceedings of the 2007 IEEE International Symposium on Industrial Electronics*, pages 1986–1991, June 2007.

[12] A. Gupta, S. Paul, Q. Jones, and C. Borcea. Automatic Identification of Informal Social Groups and Places for Geo-Social Recommendations. In *the International Journal of Mobile Network Design and Innovation. To Appear.* 2008.

[13] J. P. Sousa and D. Garlan. Aura: an architectural framework for user mobility in ubiquitous computing environments. In *3rd Working IEEE/IFIP Conference on Software Architecture*, pages 29–43. Kluwer Academic Publishers, Aug 2002.

[14] M. Roman, C. Hess, R. Cerqueira, R. H. Campbell, and K. Nahrstedt. Gaia: A middleware infrastructure to enable active spaces. In *IEEE Pervasive Computing Magazine*, volume 4(1). Oct-Dec 2002.

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

[16] A. K. Dey, D. Salber, and G. D. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. In *Human-Computer Interaction*, volume 16(2-4), pages 97–166. 2001.

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

[18] P. Bellavista, A. Corradi, and L. Foschini. 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.

[19] P. Bellavista, A. Corradi, and E. Magistretti. Comparing and Evaluating Lightweight Solutions for Replica Dissemination and Retrieval in Dense MANETs. In *Proceedings of the 9th IEEE International Symposium on Computers and Communications (ISCC'05)*. IEEE Computer Society Press, Jul 2005.

[20] C. Borcea, C. Intanagonwiwat, P. Kang, U. Kremer, and L. Iftode. Spatial Programming using Smart Messages: Design and Implementation. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS)*, pages 690–699, March 2004.

[21] O. Riva. Contory: A Middleware for the Provisioning of Context Information on Smart Phones. In *Proceedings of the 7th ACM International Middleware Conference (Middleware)*, pages 219–239. Springer, 2006.

[22] Lovegety: Love Japanese Style. http://www.wired.com/news/culture/0,1284,12899,00.html.

[23] 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.

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

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

[26] W. Yang, C. Cheng, and J. Dia. A location aware recommender system for mobile shopping environments. In *Expert Systems with Applications: An International Journal*, volume 34, pages 437–445. 2008.

[27] H. Heijden, G. Kotsis, and R. Kronsteiner. Mobile recommendation systems for decision making on the go. In *Proceedings of International Conference on Mobile Business*, July 2005.

[28] 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.

[29] SmartCampus (2005). http://smartcampus.njit.edu.